НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___”_____2020р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення розподілених систем»

спеціальності 121 «Інженерія програмного забезпечення»
на тему: “Моделювання мінімальних поверхонь на основі B-сплайну з квазіконформною заміною параметру”

Виконав:
студент IV курсу, групи ТВ-61
Данько Юрій Андрійович _____

Керівник:
ст. викладач
Гурін Артем Леонідович _____

Рецензент: _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту
Даньку Юрію Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Моделювання мінімальних поверхонь на основі B-сплайну з квазіконформною заміною параметру”

керівник роботи ст. викладач Гурін Артем Леонідович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від від ”25” травня 2020р.
№ **1168-с**

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи мови програмування C#, JS, платформа .Net Core 3.1, технологія ASP.Net Core MVC 3.1
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати методи побудови мінімальних поверхонь, обґрунтувати обрані технології для програмної реалізації, створити алгоритмічну базу для реалізації побудови мінімальних поверхонь обраними технологіями, розробити програмний продукт для побудови мінімальних поверхонь на основі вхідних даних.
5. Перелік ілюстративного матеріалу
1. Визначення мінімальних поверхонь. 2. Мінімальні поверхні у природі. 3. Проблеми сучасних інженерних рішень. 4. B-сплайни. 5. Алгоритм Кокса-де Бура. 5. Ізотропні криві. 6. Загальний алгоритм побудови об’єктів дійсного простору на основі

ізотропних характеристик. 7. Моделювання дійсних мінімальних поверхнонь методом деформації плоских ізотропних B-сплайнів з квазіконформною заміною параметру. 8. Стек технологій для програмної реалізації. 9. Архітектура розробленої системи. 10. Інформаційні потоки розробленої системи. 11. Діаграма прецедентів. 12. UML-діаграма основних класів. 13. Приклад роботи системи. 14. Висновки.

6. Дата видачі завдання 11 жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	11.10.2019	
2.	Вивчення та аналіз задачі	11.10.2019 - 23.12.2019	
3.	Розробка архітектури та загальної структури системи	03.02.2020 - 04.03.2020	
4.	Розробка архітектури окремих підсистем	05.03.2020 - 12.04.2020	
5.	Програмна реалізація системи	13.04.2020 - 17.05.2020	
6.	Оформлення пояснювальної записки	18.05.2020- 05.06.2020	
7.	Захист програмного продукту	10.06.2020	
8.	Передзахист	10.06.2020	
9.	Захист	16.06.2020	

Студент _____ Данько Ю. А.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Гурін А. Л.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Пошук мінімальних поверхонь є актуальною задачою у наш час. З розвитком сучасних комп’ютерів зростає попит на програмне забезпечення для графічних дизайнерів, архітекторів, науковців для яких мінімальні поверхні є важливими.

Мета роботи – проаналізувати метод побудови мінімальних поверхонь на основі B-сплайнів з квазіконформною заміною параметру та розробити програмний продукт для побудови мінімальної поверхні на основі початкових вхідних даних.

Записка містить 70 сторінок, 44 рисунки, 3 додатки та 13 посилань.

Ключові слова: МІНІМАЛЬНА ПОВЕРХНЯ, B-СПЛАЙНИ, КВАЗІКОНФОРМНА ЗАМІНА ПАРАМЕТРУ, АЛГОРИТМ КОКСА-ДЕ-БУРА.

ABSTRACT

Finding minimal surfaces is an urgent task nowadays. With the development of modern computers, there is a growing demand for software for graphic designers, architects, scientists for whom minimal surfaces are important.

The purpose of the work is to analyze the method of construction of minimal surfaces based on B-splines with quasi-conformal parameter change and to develop a software product for construction of a minimum surface based on initial input data.

The note contains 70 pages, 44 figures, 3 attachments and 13 links.

Keywords: MINIMUM SURFACE, B-SPLINES, QUASI-CONFORMED PARAMETER REPLACEMENT, COX-DE-BURG ALGORITHM.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

MVC – Model-View-Controller (модель-представлення-контролер).

JS - Javascript(мультипарадигмова мова програмування).

JSON - Javascript Object Notation(текстовий формат обміну даними, заснований на javascript)

ВСТУП

У час активного розвитку ком’ютерних технологій попит на програмне забезпечення для роботи з графікою необмежено зростає. Оскільки спосіб представлення інформації в графічному вигляді є наочним, тому вона сприймається людиною краще. Інформаційні дані подані в концентрованому вигляд розвивають уяву, дозволяють абстрактно мислити, надають можливість гнучко оцінювати ситуацію при зміні вхідних даних, дозволяє генерувати нові ідеї, позбутися від стереотипів і творчо підходити до обговорення певної предметної області. Тому застосування графічних програмних продуктів у виробництві, наукової та освітньої діяльності є актуальним і важливим завданням для фахівців в сфері інформаційних технологій.

В математиці, мінімальна поверхня є поверхнею, яка локально мінімізує її площу. Термін «мінімальна поверхню» використовуються тому, що ці поверхні спочатку виникли як поверхні, які зведені до мінімуму загальної площі поверхні предмета до деякого обмеження. Прикладами таких поверхонь можуть служити мильні бульбашки (різниця тисків відмінна від нуля, середня кривизна постійна і відмінна від нуля) і мильні плівки, що затягують дротові контури (тиску однакові, середня кривизна дорівнює нулю)

Мінімальні поверхні є невід’ємною частиною інструментів, використовуваних сучасними дизайнерами. В архітектурі спостерігається великий інтерес до розтягуючих структурам, які тісно пов'язані з мінімальними поверхнями.

Мінімальні поверхні стали областю інтенсивного наукового дослідження, особливо в галузі молекулярної інженерії і матеріалознавства. Ендоплазматичний ретикулум, важлива структура в клітинної біології, пропонується під еволюційним тиском, щоб відповідати нетривіальною мінімальної поверхні.

Мінімальні поверхні також мають застосування в загальній теорії відносності.

Вони знайшли своє застосування при досліджені чорних дір та проблеми Плато.

Мета даної дипломної роботи дослідити метод побудови мінімальних поверхнонь на основі B-сплайну з квазіконформною заміною параметру та створити програмний продукт за допомогою якого, користувач матиме можливість на основі початкових вхідних даних побудувати необхідну для нього мінімальну поверхню.

Актуальність вирішення описаних завдань полягає у можливості відображати та досліджувати мінімальну поверхню з різними вхідними даними.

Зміст розділів пояснювальної записки наступний:

Перший розділ описує постановку задачі, її мету, об’єкт і предмет дослідження, проблеми, які вирішуються реалізованим програмним забезпеченням.

Другий розділ містить огляд існуючих програмних рішень.

Третій розділ містить опис методу побудови мінімальної поверхні, використовуючи B-сплайни. Даний розділ надає теоретичну основу для створення алгоритмічної бази.

Четвертий розділ описує перелік обраних засобів програмної реалізації та обґрунтування, чому саме такі засоби були вибрані для вирішення поставленої задачі.

П’ятий розділ містить опис програмної реалізації системи, зокрема опис реалізованих алгоритмів, складових програмних модулів та їх взаємодію.

У шостому розділі пояснюється методика роботи користувача з програмним продуктом та описується сценарії варіантів використання системи.

Потенційними користувачами розробленої програмної системи можуть бути професійні графічні інженери, архітектори, науковці, студенти.

1. ПОСТАНОВКА ЗАДАЧІ ПОБУДОВИ МІНІМАЛЬНОЇ ПОВЕРХНІ НА ОСНОВІ B-СПЛАЙНУ З КВАЗІКОНФОРМНОЮ ЗАМІНОЮ ПАРАМЕТРУ

Метою дипломної роботи є дослідження методу побудови мінімальних поверхнонь на основі B-сплайну з квазіконформною заміною параметру та створення програмного продукту за допомогою якого, користувач матиме можливість на основі початкових вхідних даних побудувати необхідну для нього мінімальну поверхню.

Об’єктом дослідження є процеси генерації мінімальної поверхні, використовуючи ізотропні B-сплайни четвертого порядку.

Предметом дослідження є програмне забезпечення для реалізації процесу побудови мінімальної поверхні.

Для розв’язання поставленої задачі, були сформовані наступні завдання:

- проаналізувати метод побудови мінімальної поверхні;
- обрати інструменти для програмної реалізації;
- створити структуру програмного продукту;
- створити алгоритмічну базу для розрахунку координат точок мінімальної поверхні
- розробити програмний продукт для побудови мінімальної поверхні на основі початкових вхідних даних.

Завдання, що описані вище, будуть розв’язані на основі методів аналітичної геометрії та методів деформації плоских сіток.

Для роботи системи необхідна наступна вхідна інформація:

- дійсні частини координат контрольних точок характеристичного багатокутника;
- уявні частини координати першої контрольної точки характеристичного багатокутника ;
- вузловий вектор.

Вихідною інформацією є:

· побудована на основі вхідної інформації мінімальна поверхня

· характеристичні коефіцієнти мінімальної поверхні для точок, що належать поверхні

Програмна система повинна вирішувати наступні задачі:

1. Розрахунок координат поверхні на основі вхідної інформації
2. Розрахунок характеристичних коефіцієнтів для точок поверхі
3. Графічне відображення розрахованих координат
4. Вивід розрахованих характеристичних коефіцієнтів на екран користувача

Програмний продукт розроблено мовою програмування C# під платформу .Net Core, використовуючи технологію ASP.Net Core MVC, в середовищі Rider 2020 з використанням мови програмування JS та бібліотеки Plotly.js.

2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

Сучасному інженеру в наш час необхідні інструменти для вирішення математичних та геометричних задач, які перед ним постають. Для математичних розрахунків існують такі системи як Mathcad та Matlab. Для геометричних задач існують такі системи автоматизованого проектування(CAD) як AutoCAD та Autodesk Inventor. Математичні системи надають потужні можливості для складних розрахунків, а CAD-системи - для моделювання геометричних об’єктів, але не є можливим використовувати такі системи для побудови поверхонь з певними характеристиками, наприклад ізотропними, тому доцільно провести аналіз існуючих інструментів та розробити систему, яка зможе працювати з комплексними числами та будувати поверхні

2.1 Mathcad

Mathcad - це додаток для математичних і інженерних обчислень, промисловий стандарт проведення, поширення та зберігання розрахунків. Mathcad - продукт компанії PTC - світового лідера розробки систем САПР, PDM і PLM. Mathcad є універсальною системою, тобто може використовуватися в будь-якій області науки і техніки - скрізь, де застосовуються математичні методи.

Документи Mathcad представляють розрахунки у вигляді, дуже близькому до стандартного математичного мови, що спрощує постановку і рішення задач. Mathcad містить текстовий і формульний редактор, обчислювач, засоби наукової і ділової графіки, а також величезну базу довідкової інформації, як математичної, так і інженерної. Редактор формул забезпечує природний «багатоповерховий» набір формул в звичній математичній нотації (ділення, множення, квадратний корінь,

інтеграл, сума і т.д.). Потужні засоби побудови графіків і діаграм поєднують простоту використання і ефектні способи візуалізації даних і підготовки звітів.

Додаток PTC Mathcad має зручний у використані інтерфейс, з природним математичним представленням і інтелектуальним керуванням одиниць виміру. Що найважливіше, обчислювальні можливості програми забезпечують набагато більш точні результати, ніж електронні таблиці. Використання багатого набору математичних функцій PTC Mathcad дозволяє документувати найважливіші інженерні розрахунки так само просто, як робити записи в блокноті. За допомогою цієї програми можна показати свою роботу за допомогою широких можливостей форматування, а також діаграм, тексту і зображень в єдиному професійно відформатированном документі. Щоб створювати та використовувати дані в форматі PTC Mathcad, спеціалізовані навички не потрібні.

Інструменти обчислювальння Mathcad забезпечують обчислення за допомогою складних математичних формул, включаючи числові методи та аналітичні перетворення. Mathcad має великий набір вбудованих математичних функцій, дозволяє обчислювати ряди, суми, продукти, інтеграли, похідні, працювати зі складними числами, вирішувати лінійні та нелінійні рівняння, а також диференціальні рівняння та системи, мінімізувати та максимізувати функції, виконувати векторні та матричні операції. статистичний аналіз тощо. Контроль і перерахунок розмірів у різних вимірювальних системах здійснюється автоматично.

На рисунку 2.1 зображено приклад використання Mathcad.

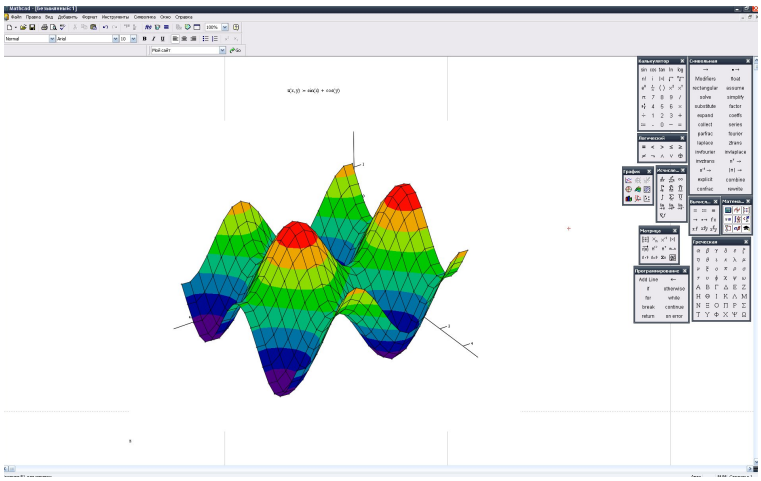

Рисунок 2.1 Приклад використання Mathcad

2.2 Matlab

MATLAB - пакет прикладних програм, призначених для роботи над технічними задачами. Пакет використовують в інженерних та наукових сферах. Підтримується більшістю сучасних операційних систем таких як Windows, Linux, macOS.

Мова MATLAB є високорівневою мовою програмування, яка має структури даних, що засновані на матрицях, широкий спектр функцій, інтегроване середовище розробки, об'єктно-орієнтовані можливості та зручні інтерфейси до програм. Програми, написані на MATLAB, є двох типів, а саме: у вигляді функцій або скриптів. Функції мають вхідні і вихідні аргументи, а також власний робочий простір для зберігання проміжних результатів обчислень і змінних. Скрипти ж використовують загальний робочий простір. Як скрипти, так і функції зберігаються у вигляді текстових файлів і компілюються в машинний код динамічно. Існує також можливість зберігати так звані pre-parsed програми - функції і скрипти, оброблені в вид, зручний для машинного виконання. У загальному випадку такі програми виконуються швидше звичайних, особливо якщо функція містить команди побудови графіків.

На рисунку 2.2 зображено приклад роботи в MATLAB.

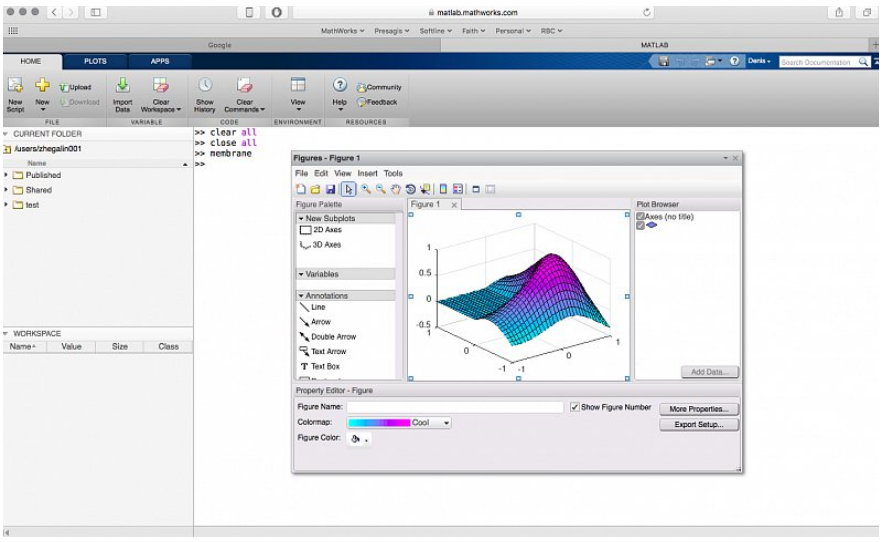

Рисунок 2.2 - Приклад роботи в MATLAB

2.3 AutoCAD

AutoCAD - це базова САПР, що розробляється і поставляється компанією Autodesk. AutoCAD - найпоширеніша CAD-система в світі, що дозволяє проектувати як в двовимірної, так і тривимірному середовищі. За допомогою AutoCAD можна будувати 3D-моделі, створювати і оформляти креслення. AutoCAD є платформною САПР, тобто ця система не має чіткої орієнтації на певну проектну область, в ній можна виконувати-будівельні, машинобудівні проекти, працювати з дослідженнями, електрикою тощо.

В області двовимірного проектування AutoCAD дозволяє використовувати елементарні графічні примітиви для отримання більш складних об'єктів. Програма надає широкі можливості роботи з шарами і аннотаційними об'єктами (розмірами, текстом, позначеннями). Використання механізму зовнішніх посилань дозволяє розбивати креслення на складові файли, за які відповідальні різні розробники, а динамічні блоки розширюють можливості автоматизації 2D-проектування звичайним користувачем без використання програмування.

На рисунку 2.3 зображено приклад роботи в AutoCAD.

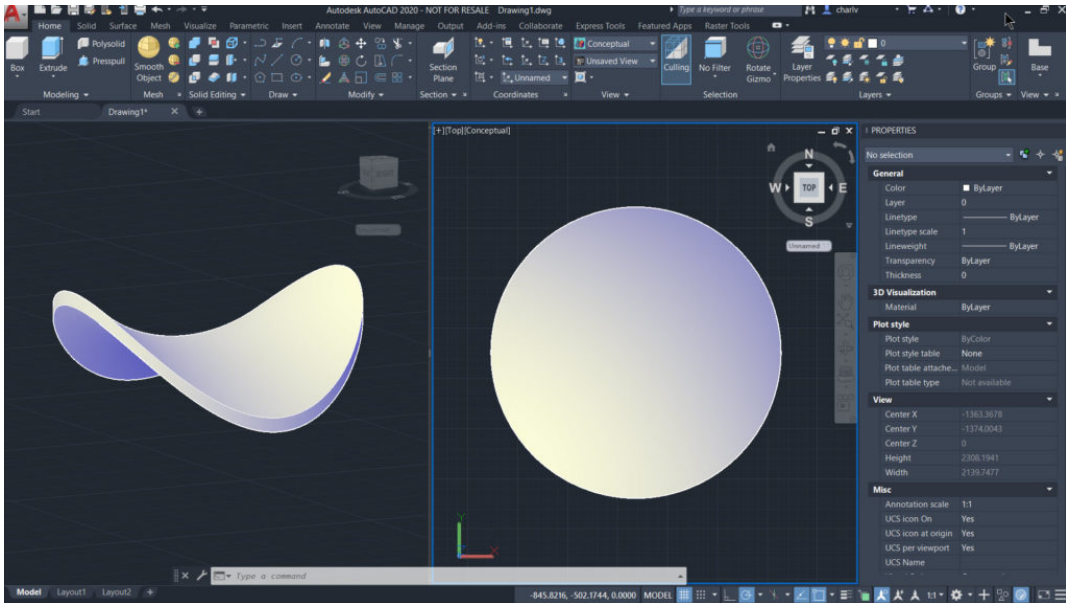

Рисунок 2.3 - Приклад роботи в AutoCAD

3. ТЕОРЕТИЧНА ОСНОВА МЕТОДІВ РОЗ'ВЯЗАННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

Мінімальні поверхні є необхідним інструментом для різних промислових галузей. В даному розділі описано теоретичне підґрунтя для моделювання мінімальних поверхонь з використанням B-сплайнів.

3.1 Параметричні криві

В параметричному вигляді кожна координата точки кривої представлена як функція одного параметру. Координатний вектор точки на кривій задає значення параметру. Координтати точки плоскої двовимірної кривої з параметром t дорівнюють

$$\begin{aligned} x &= x(t), \\ y &= y(t) \text{ [1].} \end{aligned} \tag{3.1}$$

Векторне представлення точки на плоскій кривій задається наступним чином:

$$L(t) = [x(t)\ y(t)]. \tag{3.2}$$

Щоб отримати непараметрично задану криву потрібно усунути параметр t у двох рівняннях та вивести одне зі змінними x та y [1].

Координтати точки просторової трьохвимірної кривої з параметром t дорівнюють

$$\begin{aligned} x &= x(t), \\ y &= y(t), \\ z &= z(t) \text{ [1].} \end{aligned} \tag{3.3}$$

Векторне представлення точки на просторовій кривій задається наступним чином:

$$L(t) = [x(t)\ y(t)\ z(t)] \text{ [1].} \tag{3.4}$$

Використання параметричного способу задання дозволяє представляти багатозначні та замкнуті криві.

Наслідком того, що точка на параметричній кривій визначається тільки значенням параметру, є те, що ця форма не залежить від вибору системи координат. Довжина кривої та значення координат точки визначається інтервалом зміни параметру [1].

Часто використовують нормілізацію параметру в межах від 0 до 1 на певній ділянці кривої.

3.2 Криві Безьє

Крива Безьє — це крива, що задається в параметричному вигляді, що віднайшла своє застосування в таких галузях як комп'ютерній графіці та суміжних їй областях. Просторові криві Безьє використовують для побудови поверхнонь Безьє.

При моделювання гладких кривих у векторній графіці, які можна масштабувати до нескінченності, використовуються криві Безьє. «Шляхи», відомі у програмах для роботи з зображеннями, є не чим іншим як комбінаціями з'єднаних кривих Безьє. Шляхи не обмежуються розмірами растрових зображень і їх редагування є інтуїтивно зрозумілим. Криві Безьє застосовуються в анімації в ролі інструменту для управління рухом та системах автоматизованого проектування.

Крива Безьє — параметрична крива, вигляду

$$B = \sum_{i=0}^{n} b_{i,n}(t) * P_i, \qquad t \in [0,1] \tag{3.5}$$

де P_i - опорні вершини,

$$b_{i,n}(t) = \binom{n}{i} * t^i * (1-t)^{n-i}, \tag{3.6}$$

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}, \tag{3.7}$$

де $b_{i,n}(t)$ - поліном Бернштейна, вони є базисними функціями кривої Безьє [2].

Крива Безьє задається багатокутником, як показано на рисунку 3.1. Беручи в увагу те, що базис Безьє є бернштейновскім, відразу ж відомі деякі властивості кривих Безьє [2], наприклад:

1. Значення базисних функцій є дійсними числами
2. Ступінь многочлена, що визначає ділянку кривої, є меншим кількості точок відповідного багатокутника рівно на одиницю
3. Перша і остання точки кривої збігаються з відповідними точками багатокутника
4. Основа форми кривої повторює обриси багатокутника
5. Вектори дотичних в кінцях кривої у напрямку збігаються з першої і останньої сторонами багатокутника
6. Крива має властивість зменшення варіації. Мається на увазі те, що крива перетинає будь-яку пряму лінію не частіше, ніж визначальний багатокутник. Крива є інваріантною до афінних перетворень
7. Крива лежить всередині опуклої оболонки багатокутника, тобто усередині найбільшого багатокутника, побудованого на основі заданих точок.

На рисунку 3.1 опукла оболонка позначена штриховими лініями

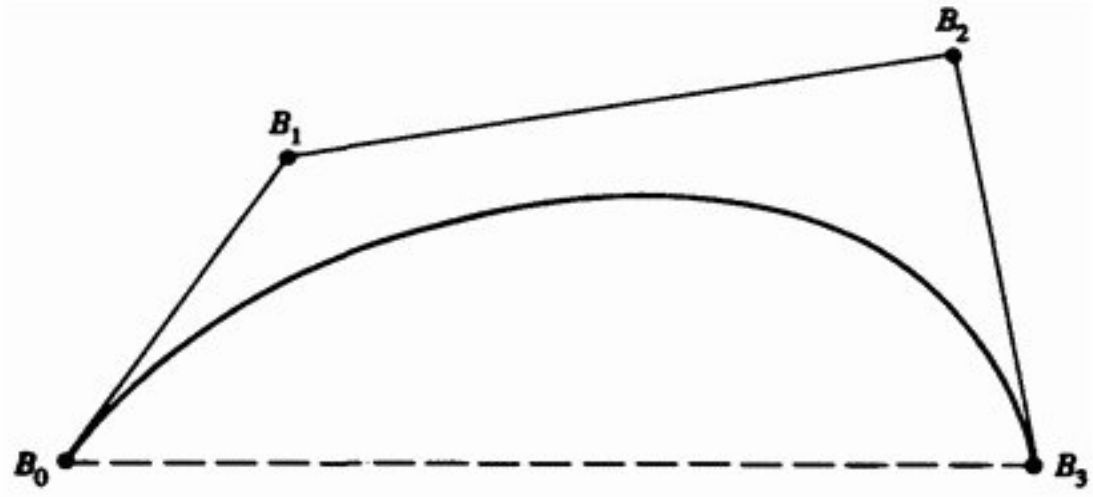

Рисунок 3.1 – Крива Безьє і її визначні точки

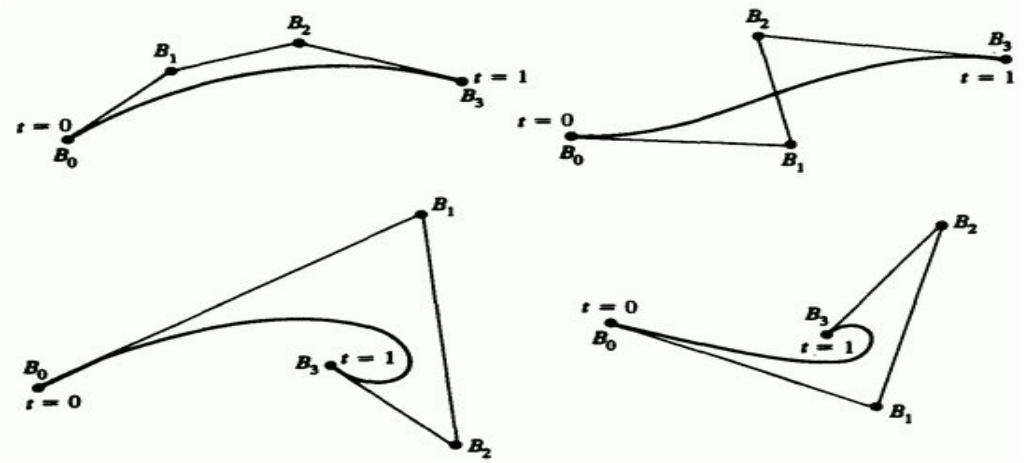

Рисунок 3.2 – Багатокутники Безьє для кубічних кривих

3.3 B-сплайни

Розглядаючи криву, яка задається вершинами багатокутника, з математичної точки зору можна зробити висновок, що вона залежить від інтерполяції або апроксимації, які встановлюють зв'язок кривої і багатокутника. В даному випадку основою є вибір базисних функцій. Криві Безьє породжуються базисом Бернштейна, який володіє двома властивостями, які обмежують гнучкість кривих. По-перше, кількість вершин багатокутника жорстко задає порядок многочлена. Наприклад, кубічна крива повинна бути задана чотирма вершинами і трьома відрізками. Шестикутник завжди породжує криву п'ятого порядку. Відомим і єдиним способом понизити ступінь кривої - це скоротити кількість вершин, а підвищити ступінь кривої - збільшити їх число. Друге обмеження є наслідком глобальної природи базису Бернштейна. Це дає право вважати, що величина аппроксимуючих функцій з рівняння 3.6 є відмінною від нуля для всього інтервалу значень параметра на кривій. Кожна точка на кривій Безьє залежить від всіх визначальних вершин, тому зміна координат однієї вершини матиме вплив на всю криву. Наслідком цього є неможливість змінювати криву локально.

Так як нахил сторін характеристичного багатокутника задає нахил відповідних кінців кривої Безье, можна пересунути середню вершину п'ятиточкового багатокутника, не змінюючи напрямку на кінцях. Через те, що базис Бернштейна має глобальний характер, форма кривої змінюється на всьому інтервалі допустимих значень. Можливість локального впливу на криву є ключовою потребую в деяких прикладних задачах.

Існує інший, неглобальний базис - базис B-сплайна, що включає в себе базис Бернштейна як окремий випадок. B-сплайни є неглобальними, оскільки кожній вершині характеристичного багатокутника відповідає своя базисна функція. Тому вплив кожної вершини на криву проявляється тільки в тих випадках, коли відповідна базова функція не дорівнює нулю при вхідних значенніх параметру. Базис B-сплайна також надає можливість змінювати порядок базисних функцій. Наслідком цього є допустимий сценарій зміни всієї кривої без зміни кількості вершин. Рекурсивне визначення для чисельного рішення було запропоновано незалежно Коксом і де Буром [3].

Нехай $P(t)$ визначає криву як функцію від параметра t, тоді B-сплайн має вигляд

$$P(t) = \sum_{i=0}^{n} B_i N_{i,k}, \quad t_{min} \le t \le t_{max}, \text{ де} \tag{3.8}$$

B_i - вершина характеристичного багатокутника, а $N_{i,k}$ - нормалізована функція базиса B-сплайна.

Для i-ї нормалізованої функції базиса порядку k (степеня $k - 1$) функції базиса $N_{i,k}(t)$ визначаються рекурсивними формулами Кокса-де Бура:

$$N_{i,0} = \begin{cases} 1, \text{ якщо } x_i \le t < x_{i+1} \\ 0, \text{ інакше} \end{cases} \tag{3.9}$$

$$N_{i,k} = \frac{t - x_i}{x_{i+k} - x_i} * N_{i,k-1}(t) + \frac{x_{i+k+1} - t}{x_{i+k+1} - x_{i+1}} * N_{i+1,k-1}(t). \tag{3.10}$$

Величини x_i - це елементи вузлового вектору, що задовольняють умову

$x_i \le x_{i+1}$.

Параметр t змінюється від t_{min} до t_{max} вздовж кривої $P(t)$. Вважається $0/0 = 0$ [4].

Визначення формули Кокса-де Бура (3.10) є рекурсивним. Звідси слідує, що функція порядку k залежить від базисних функцій, порядок яких менший. Нехай $N_{i,k}(t)$ - базисна функція. Для пошуку базисних функцій зручно користуватися наступною залежністю (Рисунок 3.3) [5].

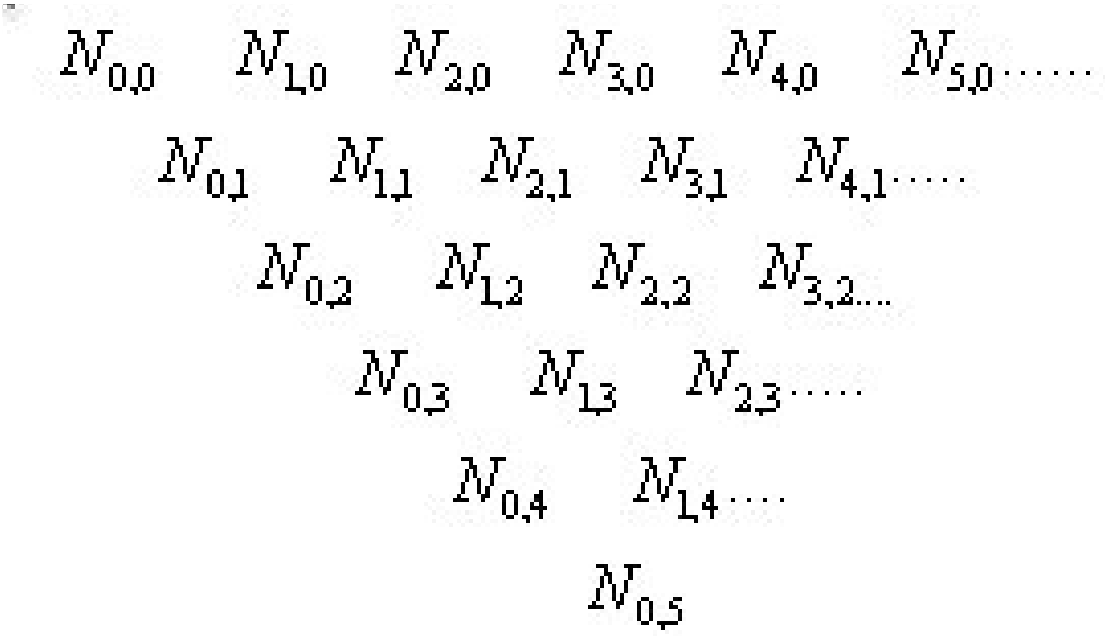

Рисунок 3.3 - Залежність між базисними функціями, подана у вигляді трикутника

Наприклад, для того, щоб знайти значення базисної функції $N_{1,2}(t)$, необхідно визначити значення базисних функцій більш низького порядку:

$$N_{1,0}(t), N_{2,0}(t), N_{3,0}(t) \rightarrow N_{1,1}(t), N_{2,1}(t) \rightarrow N_{1,2}(t).$$

B-сплайн визначається як поліноміальний сплайн порядку k (степеня $k - 1$), оскільки задовольняє тій умові, що функція $P(t)$ є поліномом $k - 1$ степеня на кожному з інтервалів $[x_i, x_{i+1})$.

Наприклад B-сплайн четвертого порядку є не чим іншим, як кусочно-кубічною кривою.

Через те, що B-сплайн задається базисом B-сплайн, слідує ще кілька властивостей:

1. При будь-якому значенні параметра t сума базисних функцій дорівнює одиниці;

$$\sum_{i=0}^{n} N_{i,k}(t) = 1, \quad (3.11)$$

2. Кожна базисна функція $N_{i,k}(t)$ є невід'ємною для всіх значеннях параметру t;
3. Максимальний порядок B-сплайну дорівнює кількості вершин характеристичного багатокутника
4. Характеристичний багатокутник задає форму кривій
5. B-сплайн лежить всередні випуклої оболочки характеристичного багатокутника
6. B-сплайн перетинає будь-яку криву не частіше за характеристичний чотирикутник
7. Щоб застосувати аффінне перетворення до B-сплайн, необхідно застосувати його до вершин характеристичного багатокутника

Вибір вузлового вектору має значний вплив на значення базисних функцій $N_{i,k}(t)$ B-сплайну і на сам B-сплайн відповідно. Вимогою до вузлового вектору є те що він повинен бути монотонно зростаючою послідовністю дійсних чисел. Існує три типи вузлових векторів:

1. Рівномірні;
2. Відкриті;
3. Нерівномірні;

Елементи рівномірного вузлового вектору розташовані на однаковій відстані один від одного, наприклад:

$$[0,1,2,3,4,5,6]$$

$$[-0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, 0.4]$$

Зазвичай перший елемент рівномірного вузлового вектору дорівнює нулю і наступні елементи збільшуються на одиницю і прямують до деякого максимального значення або нормуються десятковими значеннями в діапазоні від 0 до 1, наприклад:

$$[0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1]$$

Для порядку k рівномірні вузлові вектори породжують рівномірні періодичні функції базису для яких

$$N_{i,k}(t) = N_{i-1,k}(t - 1) = N_{i+1,k}(t + 1).$$

З цього випливає, що кожна функція базису є паралельним переносом іншої функції.

У відкритого рівномірного вузлового вектору кількість однакових вузлових значень в кінцях дорівнює порядку базисної функції B-сплайну. Значення внутрішніх елементів розподілені рівномірно [6]. Наприклад:

$$\begin{aligned} k = 2 \qquad & [0, 0\ , 1, 2, 3, 4, 4\ ], \\ k = 3 \qquad & [0, 0, 0, 1, 2, 3, 3, 3], \\ k = 4 \qquad & [0, 0, 0, 0, 1, 2, 2, 2, 2]. \end{aligned}$$

Відкритий вузловий вектор визначається як:

$$\begin{aligned} x_i &= 0 \quad 0 \le i \le k - 1 \\ x_i &= i - k \quad k \le i \le n \\ x_i &= n - k + 2 \quad n + 1 \le i \le n + k\,. \end{aligned} \tag{3.12}$$

Базисні функції в даному випадку приблизно мають таку ж поведінку як і криві Безьє. Якщо кількість вершин характеристичного багатокутника дорівнює порядку базиса B-сплайна і до того ж використовується відкритий вузловий вектор, тоді базис B-сплайна переходить у базис Бернштейна. Звідси витікає, що даний B-сплайн є кривою Безьє [7]. Вузловий вектор будет містити на початку k нулів за якими йдуть k одиниць. Наприклад, якщо характеристичний чотирикутник є п’ятикутником, тоді вузловий відкритий рівномірний вектор матиме наступний вигляд:

$$[0, 0\ , 0, 0, 0, 1, 1, 1, 1, 1\ ].$$

У відкритого B-сплайну будь-якого порядку перша та остання точки кривої збігаються з відповідними точками характеристичного багатокутника. Нахил кривої в першій та останній вершинах характеристичного багатокутника дорівнює нахилу відповідних сторін характеристичного багатокутника.

Відмінність нерівномірних вузлових векторів полягає в тому, що значення внутрішніх вузлових елементів розташовуються на різній відстані один від одного або збігаються.

Змінювати форму B-сплайну можна різними способами:

1. Змінюючи порядок k базисних функцій.
2. Використовувати повторно вершини характеристичного багатокутника.
3. Використовувати повторно вузлові значення, що розміщенні в узловому векторі.
4. Змінювати тип вузлового вектору і базису: нерівномірний, відкритий рівномірний, періодичний рівномірний.
5. Змінювати кількість і розташування точок характеристичного багатокутника.

3.4 Моделювання плоских сіток на основі ізотропних B-сплайнів

Ізотропні криві - це криві з нульовою довжиною. Оскільки в дійсному евклідовому просторі побудувати таку криву неможливо, тому використовують комплексний простір. В комплексному просторі криві з нульовою довжиною називають ізотропними кривими.

Визначною особливістю плоских, а також просторових ізотермічних сіток є форма їх елементерних комірок, що є квадратами.

Практичне застосування ізотермічні сітки віднайшли в моделюванні поширення тепла та побудови мінімальних поверхонь.

$s(t)$ - певна ізотропна крива. Для проектування ізотропних кривих використовується наступний підхід, а саме: задання дісних координат точок характеристичного багатокутника та на основі ізотропності визначення уявних частин комплексної координати характеристичного багатокутника [8].

$$s_{jRe} \rightarrow s_{jIm} \tag{3.13}$$

Ізотропну криву використовують як напрямну криву для побудови сіток.

Для цього необхідно провести комплексну заміну, а саме квазіконформну [8]:

$$t = u + ikv. \tag{3.14}$$

В залежності від способу задання вузлового вектору можна отримати рівномірно періодичні, відкриті та нерівномірні сплайни.

Періодичний В-сплайн четвертого порядку з нормалізованою параметризацією є не чим іншим як сукупністю сегментів, базисні функції яких є в межах інтервалу параметру [8], а саме

$$0 \le t \le 1, \tag{3.15}$$

та є однаковими.

$$s_j(t) = \frac{1}{6}\left[t^3, t^2, t, 1\right]\begin{bmatrix} 0 & 3 & -6 & 3 \\ 0 & 3 & 0 & -3 \\ 1 & -3 & 4 & 3 \\ -1 & 1 & 1 & 0 \end{bmatrix}\begin{bmatrix} s_j \\ s_{j+1} \\ s_{j+2} \\ s_{j+3} \end{bmatrix}, (0 \le t \le 1) \tag{3.16}$$

де j - кількість сегментів кривої В-сплайну,

$s_j, s_{j+1}, s_{j+2}, s_{j+3}$ - точки характеристичного чотирикутника.

Сегмент кривої періодичного В-сплайну четвертого порядку має наступний аналітичний опис [8]:

$$\begin{aligned} s_j(t) = \frac{1}{6}(s_j(-t^3 + 3t^2 - 3t + 1) + \\ s_{j+1}(3t^3 - 6t^2 + 4) + s_{j+2}(-3t^3 + 3t^2 + 3t^2 + 3t + 1) + s_{j+1}t^3) \end{aligned} \tag{3.17}$$

Точки характеристичного чотирикутника задаються у комплесному вигляді:

$$s_{j+1} = [Re(x_{j+1}) \pm iIm(x_{j+1})\, Re(y_{j+1}) \pm iIm(y_{j+1})] \tag{3.18}$$

Для того, щоб побудувати періодичний В-сплайн необхідно відокремити дійсну частину, при цьому уявна частина не матиме впливу на побудову кривої.

Щоб знайти невідомі уявні частини точок характеристичного чотирикутника необхідно скористатися означенням ізотропності кривої, а саме: довжина кривої на комплесній площині повинна бути рівною нулю [8]. Виходячи з цього матимемо:

$$\begin{aligned} y_{j+1} &= y_j + \mathrm{i}(x_j - x_{j+1}), \\ y_{j+2} &= y_j + \mathrm{i}(x_j - x_{j+2}), \\ y_{j+3} &= y_j + \mathrm{i}(x_j - x_{j+3}). \end{aligned} \tag{3.19}$$

Наступним кроком є виділення дійсної та уявної частини:

$$
\begin{aligned}
Im(x_{j+1}) &= -\,Re(y_j) + Re(y_{j+1}) \;+\; Im(x_j),\\
Im(y_{j+1}) &= Re(x_j) - Re(x_{j+1}) \;+\; Im(y_j),\\
Im(x_{j+2}) &= -\,Re(y_j) + Re(y_{j+2}) \;+\; Im(x_j),\\
Im(y_{j+2}) &= Re(x_j) - Re(x_{j+2}) \;+\; Im(y_j),\\
Im(x_{j+3}) &= -\,Re(y_j) + Re(y_{j+3}) \;+\; Im(x_j),\\
Im(y_{j+3}) &= Re(x_j) - Re(x_{j+3}) \;+\; Im(y_j).
\end{aligned}
\tag{3.20}
$$

Для побудови ізотермічної сітки на основі періодичного B-сплайну четвертого порядку необхідно вирази (3.14) та (3.19) підставити у рівняння (3.17) та виділити дійсну частину [8].

Наступним етапом є аналіз геометрії побудованої сітки. Для цього потрібно знайти частинні похідні від $Re(x(u + ikv))$ та $Re(y(u + ikv))$ та підставити у формулу для розрахунку першої квадратичної форми, що дозволить оцінити довжини сегментів кривих, площі областей на сітці та кути, що утворюються між кривими [8].

$$
\begin{aligned}
F_j &= x_{uj}(u,v)x_{vj}(u,v) + y_{uj}(u,v)y_{vj}(u,v),\\
E_j &= x_{uj}(u,v)^2 + y_{uj}(u,v)^2,\\
G_j &= x_{vj}(u,v)^2 + y_{vj}(u,v)^2.
\end{aligned}
\tag{3.21}
$$

Якщо $F_j = 0$, тоді можна робити висновок про те, що побудована сітка є ортогональною [8].

Якщо $E_j = G_j$, тоді можна робити висновок про те, що побудована сітка є ізотермічною [8].

Загальний алгоритм побудови об’єктів дійсного простору на основі ізотропних характеристик зображено на рисунку 3.4 [9].

Рисунок 3.4 - Алгоритм побудови об'єктів дійсного простору на основі ізотропних характеристик

3.5 Моделювання дійсних мінімальних поверхонь на основі деформації плоских ізотропних B-сплайнів

Мінімальні поверхні - це поверхні, у яких середня кривизна в будь-якій її точці дорівнює нулю.

Геометрична форма мінімальної поверхні забезпечує рівномірний розподіл зусиль в оболонці та додаткову жорсткість.

Рівність нулю середньої кривизни мінімальної поверхні є необхідною умовою мінімальності площі відсіку поверхні.

Для пошуку аналітичного опису мінімальних поверхонь, використовуючи функції комплесної змінної, потрібно відшукати параметричні рівняння ізотропних кривих нульової довжини.

Відомим способом знаходження просторових кривих є знаходження третьої координати на основі плоскої кривої. Для цього необхідно деформувати плоску ізотропну криву (3.17) таким чином, щоб довжина кривої в комплексому просторі

залишалася незмінною. Для плоского В-сплайну всі довжини сторін характеристичного чотирикутника та хорди дорівнюють нулю. Для просторового випадку ця умова не повинна виконуватися, тому будемо змінювати точки характекристичного чотирикутника зі збереженням умови ізотропності довжини кривої [10].

Загальна кількість умов збереження ізотропності для В-сплайну n – го порядку дорівнює $(2n - 1)$. Виходячи з цієї умови можна знайти кількість невідомих координат. Кількість координат для просторового В-сплайну дорівнює $3(n + 1)$. $3(n + 1) - (2n - 1) = n + 4$ - таку кількість координати необхідно задавати [10].

Відомі координати просторової кривої задаються у наступному вигляді:

$$s_j = [x_j, y_j], z_0, j = \frac{n+3}{2} \tag{3.22}$$

Для просторового В-сплайну четвертого порядку будемо мати наступні формули для пошуку невідомих координат точок характеристичного чотирикутника [10]:

$$\begin{gathered} z_1 = z_0, \\ y_3 = \frac{-2(z_2 - z_1)^2 - (x_1 - x_0)(x_3 - x_2)}{(y_1 - y_0)} + y_2, \\ z_3 = \frac{2\,(y_2 - y_1)(z_2 - z_1)}{(y_1 - y_0)} + z_2, \\ x_3 = \frac{-(z_2 - z_1)^2 - (y_2 - y_1)^2}{(x_1 - x_0)} + x_2 \end{gathered} \tag{3.23}$$

Загальне рівняння просторового В-сплайну четвертого порядку має вигляд:

$$\begin{gathered} P(t) = \sum_{i=0}^{3} B_i N_{i,4} = \\ B_0 * N_{0,4}(t) + B_1 * N_{1,4}(t) + B_2 * N_{2,4}(t) + B_3 * N_{3,4}(t), \\ t_{min} \le t \le t_{max}, \text{ де} \end{gathered} \tag{3.24}$$

B_i - вершина характеристичного чотирикутника, яка подається у вигляді

$$B_i = [Re(x_i) \pm iIm(x_i)\, Re(y_i) \pm iIm(y_i)\, Re(z_i) \pm iIm(z_i)], \tag{3.25}$$

$N_{i,4}$ - i-та базисна функція четвертого порядку, значення якої залежить від вибору типу вузлового вектору.

Аналітичний опис періодичного просторового B-сплайну четвертого порядку має вигляд:

$$P(t) = \frac{1}{6}(B_0(-t^3 + 3t^2 - 3t + 1) + B_1(3t^3 - 6t^2 + 4) + B_2(-3t^3 + 3t^2 + 3t + 1) + B_3 t^3) \quad (3.26)$$

Рівняння мінімальної поверхні має вигляд:

$$\begin{aligned} x(t) &= \frac{1}{6}(x_0(-t^3 + 3t^2 - 3t + 1) + x_1(3t^3 - 6t^2 + 4) + x_2(-3t^3 + 3t^2 + 3t + 1) + x_3 t^3), \\ y(t) &= \frac{1}{6}(y_0(-t^3 + 3t^2 - 3t + 1) + y_1(3t^3 - 6t^2 + 4) + y_2(-3t^3 + 3t^2 + 3t + 1) + y_3 t^3), \\ z(t) &= \frac{1}{6}(z_0(-t^3 + 3t^2 - 3t + 1) + z_1(3t^3 - 6t^2 + 4) + z_2(-3t^3 + 3t^2 + 3t + 1) + z_3 t^3), \\ t &= u + ikv,\ u \in [0, 1],\ v \in [0, 1]. \end{aligned} \quad (3.27)$$

Аналогічно для плоского випадку будемо мати формули для дослідження поверхні на ортогональність та ізотермічність:

$$\begin{aligned} F_j &= x_{uj}(u,v)x_{vj}(u,v) + y_{uj}(u,v)y_{vj}(u,v) + z_{uj}(u,v)z_{vj}(u,v), \\ E_j &= x_{uj}(u,v)^2 + y_{uj}(u,v)^2 + z_{uj}(u,v)^2, \\ G_j &= x_{vj}(u,v)^2 + y_{vj}(u,v)^2 + z_{vj}(u,v)^2 \end{aligned} \quad (3.28)$$

4. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

Платформою для рішення поставлених завдань було обрано Asp.Net Core 3.1. Оскільки дана плаформа є універсальною для таких операційних систем як Windows, Linux, macOs. Було створено рішення типу AspNet.Core MVC. В якості об’єктно-орієнтованої мови програмування обрано C#. Для побудови графічного інтерфейсу застосовано мову розмітки HTML5, таблицю стилів CSS3, мову програмування JS та бібліотеку plotly.js. Даний вибір було зроблемо спираючись на актуальність технологій та велику спільноту інженерів-програмістів.

4.1 Платформа Asp.Net Core 3.1

Платформа ASP.NET Core є технологію від компанії Microsoft, призначеною для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів.

ASP.NET Core є повністю opensource-фреймворком.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget.

ASP.NET Core - це версія з відкритим кодом ASP.NET, яка працює на macOS, Linux та Windows. ASP.NET Core вперше був випущений у 2016 році і є переосмисленням більш ранніх версій ASP.NET.

ASP.NET Core розроблений для того, щоб компоненти часу, API, компілятори та мови еволюціонували швидко, одночасно забезпечуючи стабільну і підтримувану платформу для роботи додатків.

ASP.NET Core пропонує різні варіанти життєвого циклу підтримки для задоволення потреб додатка. Можна обрати довгострокову версію підтримки або запустити її з останньою версією.

4.2 Asp.Net Core MVC 3.1

ASP.NET Core MVC - це потужний фреймворк для створення веб-додатків та API, використовуючи модель дизайну Model-View-Controller.

ASP.NET Core MVC є легким, безпечним, з відкритим вихідним кодом фреймворком, оптимізованим для використання з ASP.NET Core.

ASP.NET Core MVC пропонує побудову динамічних веб-сайтів на основі моделей, що дозволяє чітко розділяти проблеми. Він дає повний контроль над розміткою, підтримує TDD-зручну розробку та використовує найновіші веб-стандарти.

4.2.1 Патерн MVC

Архітектурний шаблон Model-View-Controller (MVC) розділяє додаток на три основні групи компонентів: моделі, представлення і контролери. Ця модель допомагає досягти поділу обов’язків. Використовуючи цей шаблон, запити користувача надходять до контролера, який відповідає за роботу з моделлю для виконання дій користувача і / або отримання результатів запитів. Контролер обирає представлення для відображення користувачеві і надає йому всі необхідні дані моделі.

Таке розмежування обов'язків допомагає масштабувати додаток з точки зору складності, тому що простіше розроблювати, налагоджувати і тестувати щось конкретне (модель, уявлення або контролер), у якого є одне завдання. Складніше оновлювати, тестувати і налагоджувати код, у якого є залежності, розподілені за двома або більше з цих трьох областей. Наприклад, логіка призначеного для користувача інтерфейсу має тенденцію змінюватися частіше, ніж бізнес-логіка. Якщо код уявлення і бізнес-логіка об'єднані в одному об'єкті, об'єкт, що містить бізнес-логіку, повинен змінюватися при кожній зміні призначеного для користувача інтерфейсу. Це є причиною виникнення помилок у застосунку та потребує повторного тестування бізнес-логіки системи навіть у випадку незначних змін у графічному інтерфейсі [11].

Схема взаємодії структурних компонентів при такому підході розробки зображено на рисунку 4.1 [11].

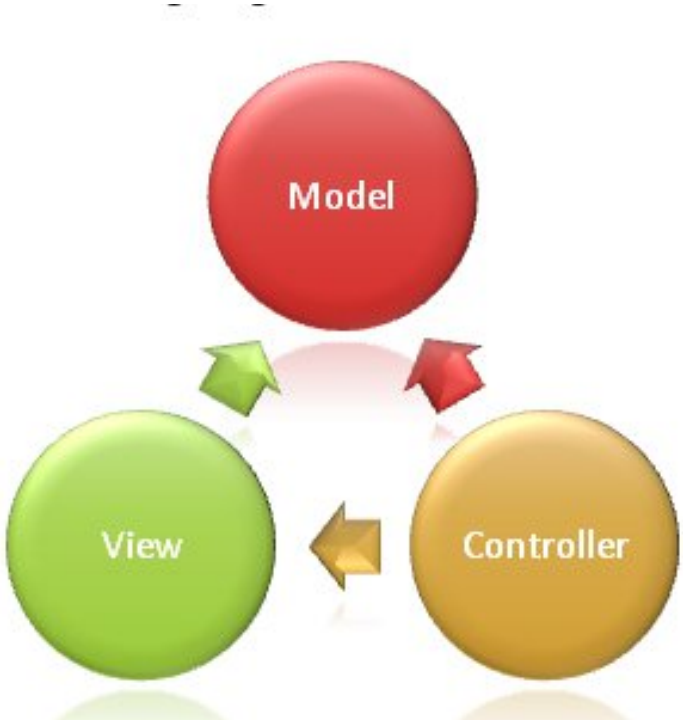

Рисунок 4.1 - Схема взаємодії компонентів в патерні проектування MVC

4.2.2 Model

Модель у додатку MVC відображає стан програми та будь-яку бізнес-логіку чи операції, які повинні виконуватись нею. Бізнес-логіка повинна бути інкапсульована у моделі разом із будь-якою логікою реалізації для збереження стану програми. Сильно типізовані представлення зазвичай використовують типи ViewModel, призначені для утримання даних, що відображаються в цьому представленні даних. Контролер створює та заповнює ці екземпляри ViewModel з моделі.

Модель надає дані і методи для роботи з ними, що задовольняють потребам користувача в певній предметній області. Модель не залежить від представлення (не знає як дані візуалізувати) і контролера (не має точок взаємодії з користувачем), просто надаючи доступ до даних і управління ними [11].

4.2.3 View

Перегляди відповідають за презентацію вмісту через користувальницький інтерфейс. Вони використовують механізм перегляду Razor, щоб вставити .NET-код у

розмітку HTML. У межах переглядів має бути мінімальна логіка, і будь-яка логіка в них повинна стосуватися подання вмісту.

У шаблоні Model-View-Controller (MVC) представлення візуалізує дані програми і відповідає за взаємодію з користувачем. Представлення - це HTML-шаблон з вбудованою розміткою Razor. Розмітка Razor - це код, який взаємодіє з розміткою HTML для створення веб-сторінки, яка надсилається клієнту.

У ASP.NET Core MVC представлення - це файли .cshtml, які використовують мову програмування C# в розмітці Razor. Зазвичай файли представлень групують в папки, які мають таку ж назву, що і контролер, який використовує їх [11].

Перегляди допомагають встановити розділення обов’язків у програмі MVC, відокремлюючи розмітку інтерфейсу користувача від інших частин програми. Такий підхід до дизайну системи робить додаток модульним, що забезпечує ряд переваг, а саме:

1. Система є легшою у підтримуванні, оскільки її архітектура краще організована. Представлення зазвичай групують за наданими системою сервісами. Це полегшує пошук пов'язаних представлень при роботі з функціоналом системи.
2. Частини програмного коду слабо пов'язані. Є можливим створювати і оновлювати графічний інтерфейс додатку незалежно від бізнес-логіки і елементів доступу до даних, тобто зміна зовнішнього вигляду системи не матиме впливу на інші частини програми.
3. Простіше тестувати частини графічного інтерфейсу користувача, оскільки представлення даних є окремими одиницями.
4. Завдяки кращій організації менш імовірно, що будуть випадково повторюватися частини призначеного для користувача інтерфейсу.

4.2.4 Controller

Контролери - це компоненти, які керують взаємодією з користувачем, працюють з моделлю і в кінцевому підсумку вибирають подання для візуалізації. У програмі MVC представлення відображає лише інформацію; контролер обробляє та

реагує на введення та взаємодію користувачів. У шаблоні MVC контролер є початковою точкою входу, і він відповідає за вибір типів моделі для роботи та який вид подати (звідси її назва - він керує тим, як програма реагує на заданий запит) [11].

Контролер використовується для визначення та групування набору дій. Дія (або метод дії) - це метод, що належить контролеру, який обробляє запити. Пов’язані за змістом дії повинні розміщуватися в одному контролері. Ця агрегація дій дозволяє застосовувати загальні набори правил, такі як маршрутизація, кешування та авторизація, застосовуватися колективн. Потрібні дії контролеру виконуються на певний запит користувача за допомогою маршрутизації.

У шаблоні Model-View-Controller контролер відповідає за первинну обробку запиту і створення екземпляра моделі. Як правило, бізнес-рішення повинні виконуватися в рамках моделі.

Контролер бере результат обробки моделі (якщо є) і повертає або підготовлене представлення з необхідними даними, або результат виклику API.

Контролер є абстракцією рівня інтерфейсу користувача. В його обов'язки входить забезпечення достовірності даних запиту і вибір того, яке представлення (або результат для API) має бути повернуто в якості результату роботи методу контролеру. В добре розроблених додатках він не включає безпосередньо доступ до даних або бізнес-логіку. Замість цього контролер делегує ці обов’язки сервісам, які інкапсулюють в собі необхідну бізнес-локіку предметної області.

4.3 Plotly.js - графічна бібліотека

Plotly - це безкоштовна бібліотека графіків з відкритим кодом для JavaScript.

Plotly.js - це бібліотека декларативних діаграм високого рівня.

Plotly.js поставляється з більш ніж 40 типами діаграм, включаючи тривимірні, статистичні і SVG-карти.

Plotly.js діаграми декларативно описуються як об'єкти JSON. Кожен аспект діаграм, наприклад кольори, лінії сітки та легенда, має відповідний набір атрибутів JSON.

Більшість сюжетних графіків намальовані за допомогою SVG. Це забезпечує велику сумісність у веб-переглядачах та експорт векторних зображень з якістю публікацій. На жаль, існують властиві обмеження продуктивності щодо кількості елементів SVG, які можна намалювати у DOM.

Plotly.js використовує stack.gl для високопродуктивних 2D і 3D графіків.

Plotly.js абстрагує типи статистичних і наукових діаграм, які ви можна знайти в таких пакетах, як matplotlib, ggplot2 або MATLAB.

Дані, які повинні бути відображені на графіку, описують в масиві, зазвичай званому даними, чиї елементи є об'єктами трасування різних типів (наприклад, розкид, стовпець тощо).

Макет графіка - візуальні атрибути, не пов'язані з даними, такі як заголовок, анотації тощо, - описується в об'єкті, зазвичай званому макетом.

Високорівневі параметри конфігурації для графіка, такі як поведінка прокрутки / масштабування / наведення, описані в об'єкті, зазвичай званому config. Різниця між конфігурацією і компонуванням полягає в тому, що компоновка відноситься до вмісту графіка, тоді як конфігурація відноситься до контексту, в якому відображається графік.

Кадри анімації описуються в об'єкті, зазвичай званому кадрами. Вони можуть містити дані і об'єкти макета, які визначають будь-які зміни для анімації, і об'єкт трасування, який визначає, які траси анімувати [12].

На рисунку 4.2 зображено приклад використання бібліотеки plotly.js для побудови ліній і графіку розсіювання.

На рисунку 4.3 зображено приклад використання бібліотеки plotly.js для побудови 3D Scatter Plot.

На рисунку 4.4 зображено приклад використання бібліотеки plotly.js для побудови топографічного 3D-графіку поверхні.

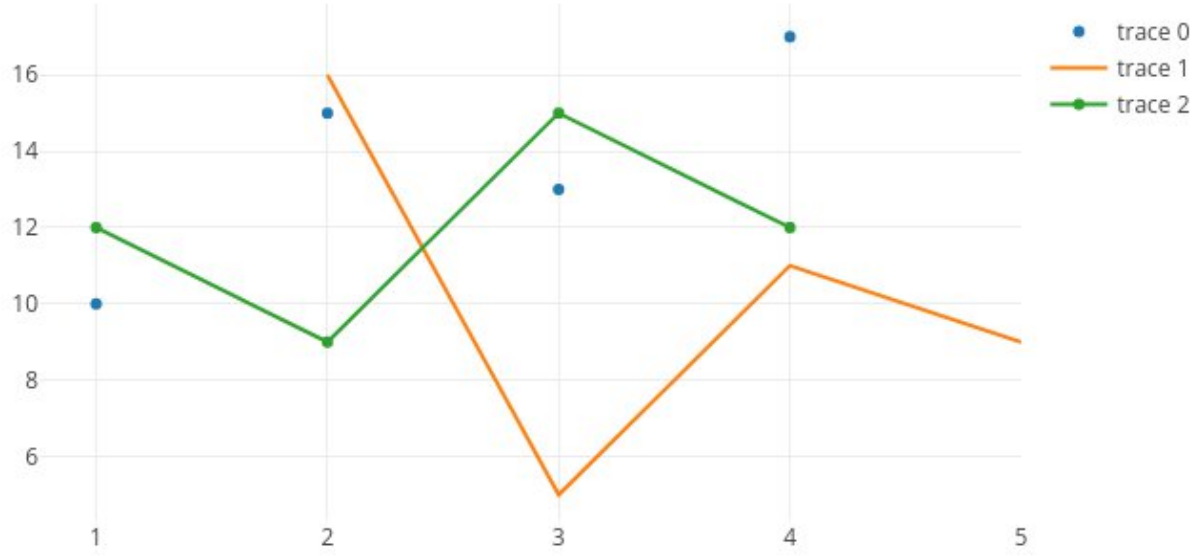

Рисунок 4.2 - Приклад використання бібліотеки plotly.js для побудови ліній і графіку розсіювання

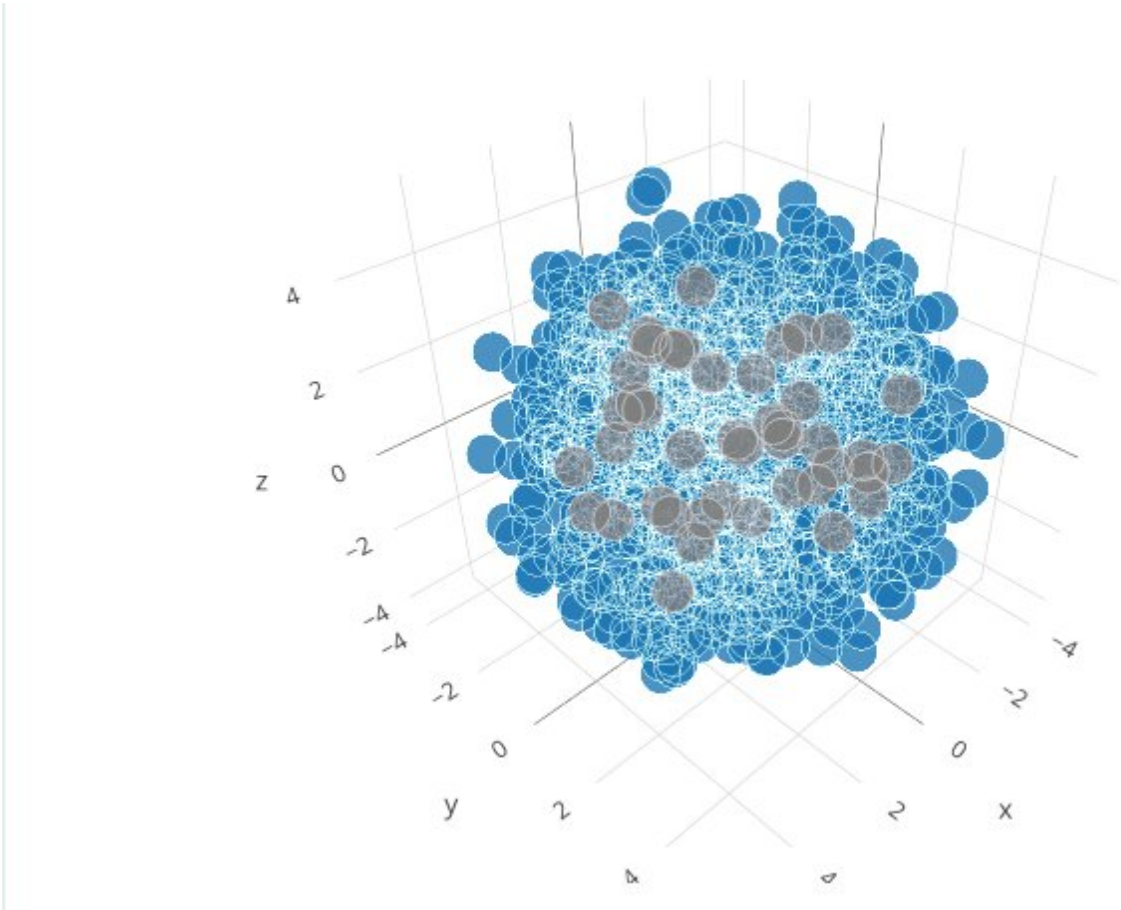

Рисунок 4.3 - Приклад використання бібліотеки plotly.js для побудови 3D Scatter Plot

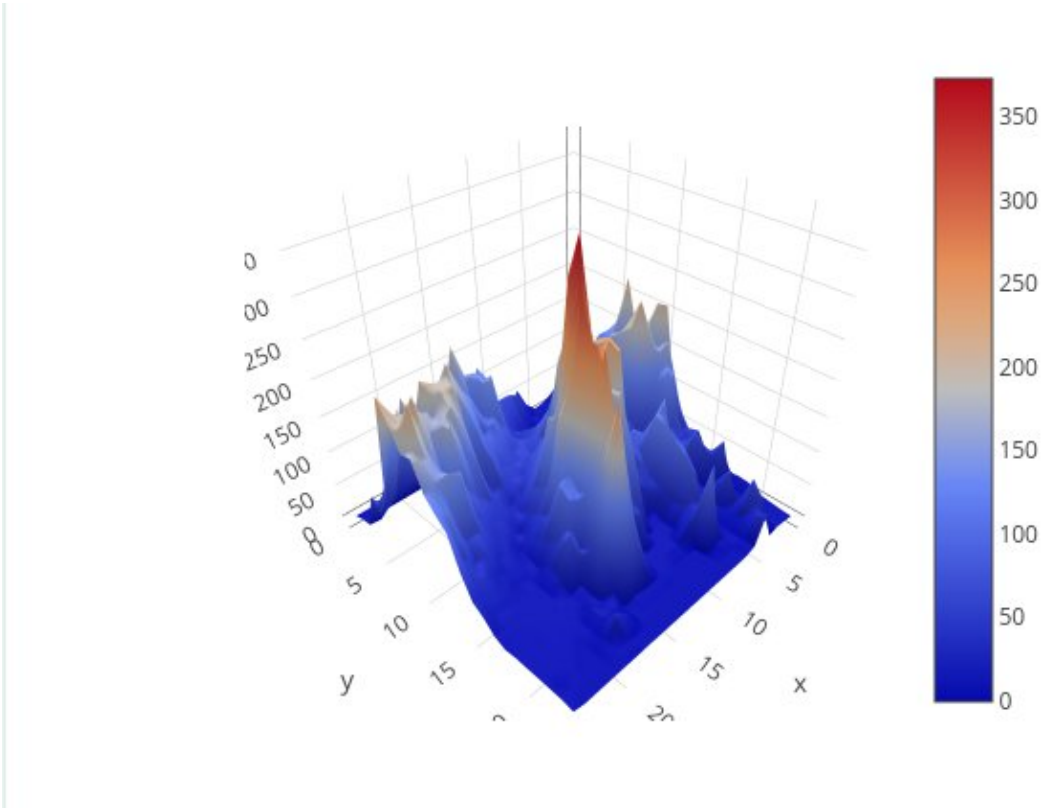

Рисунок 4.4 - Приклад використання бібліотеки plotly.js для побудови топографічного 3D-графіку поверхні

4.4 JSON файл як спосіб зберігання даних

Файл JSON - це файл, який містить дані з простою структурою та об’єкти у форматі JSON. Файли з таким розширенням є одним з можливих форматів обміну даними. Використовуються для комунікації між сервером та веб-додатком. Файли у форматі JSON зберігають текст, який людина з легкістю може читати і редагувати за допомогою будь-якого текстового редактора [13].

Формат JSON спочатку був заснований на підмножині JavaScript, але вважається незалежним від мови форматом, підтримуваним багатьма різними API-інтерфейсами програмування. JSON зазвичай використовується в програмуванні веб-додатків Ajax. Він стає все більш популярним в якості альтернативи XML [13].

Хоча багато програм використовують JSON для обміну даними, вони можуть не зберігати файли .json на жорсткому диску, оскільки обмін даними відбувається між комп'ютерами, підключеними до Інтернету. Однак деякі програми дозволяють

користувачам зберігати файли .json. Одним із прикладів є Google+, який використовує файли JSON для збереження даних профілю. Після входу в систему можна вибрати сторінку «Звільнення даних» і вибрати «Завантажити дані свого профілю».

Mozilla Firefox зберігає резервні копії закладок за допомогою файлів JSON. Файли зберігаються в каталозі профілю користувача Firefox в папці з ім'ям bookmarkbackups.

JSON заснований на двох структурах [13]:

1. Колекція пар у форматі ім'я та значення. У різних мовах програмування це реалізовано різними способами, наприклад як хеш-таблиця, словник, об'єкт, запис, структура, асоціативний масив або список ключів.
2. Упорядкований список значень. У більшості мов програмування це вектор, масив, список або послідовність.

Це універсальні структури даних. Практично всі сучасні мови програмування підтримують їх в тій чи іншій формі.

JSON має наступний синтаксис [13]:

- Об'єкти укладені у фігурні дужки ({}), їх пари ім'я-значення розділені комою (,), а ім'я та значення в парі розділені двокрапкою (:). Імена в об'єкті є рядками, тоді як значення можуть бути будь-якого з семи типів значень, включаючи інший об'єкт або масив.
- Масиви укладені в квадратні дужки ([]), а їх значення розділені комою (,). Кожне значення в масиві може бути іншого типу, включаючи інший масив або об'єкт.
- Коли об'єкти і масиви містять інші об'єкти або масиви, дані мають деревоподібну структуру.

JSON часто використовується в якості загального формату для серіалізації і десеріалізациі даних в додатках, які взаємодіють один з одним в мережі Інтернет. Ці додатки створюються з використанням різних мов програмування і працюють в різноманітних середовищах. JSON підходить для цього сценарію, тому що це відкритий стандарт, його легко читати і писати, і він більш компактний, ніж інші способи представлення інформації.

Уявлення JSON зазвичай більш компактні, ніж уявлення XML, оскільки в JSON немає закриваючих тегів. На відміну від XML, JSON не має загальноприйнятої схеми для визначення і перевірки структури даних JSON.

Приклад подання даних у форматі JSON зображено на рисунку 4.5.

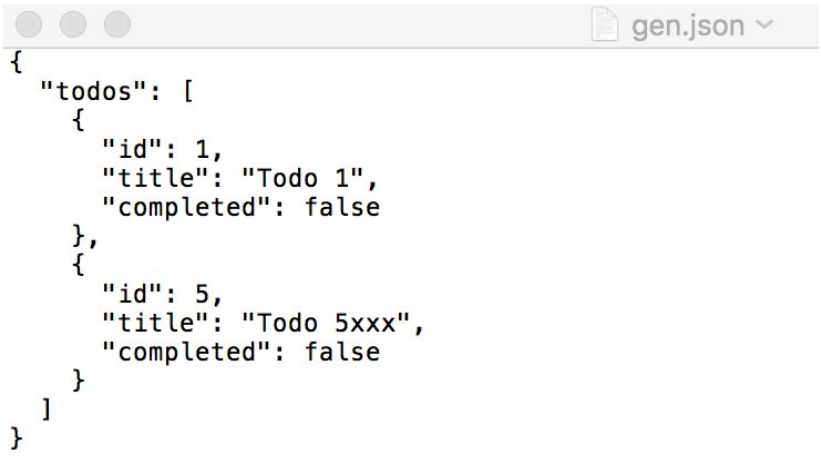

Рисунок 4.5 - Приклад подання даних у форматі JSON

4.5 Rider - крос-платформне середовище розробки для .NET

Rider - це крос-платформна IDE для .NET-розробників, заснована на платформі IntelliJ і ReSharper [14].

Переваги даного середовища розробки над іншими проявилися у:

1. Підтримці різних .NET-проектів. Rider підтримує .NET Framework, нову платформу .NET Core і проекти на основі Mono. IDE дозволяє розробляти десктопні програми, .NET-сервіси та бібліотеки, ігри на движку Unity, мобільні додатки Xamarin, веб-додатки ASP.NET і ASP.NET Core [14].
2. Швидкості роботи та наявного функціоналу. Rider надає більше 2200 інспекцій коду, сотні тематичних дій і рефакторингу, запозичених з ReSharper, в поєднанні з просунутою функціональністю середовищ

розробки на основі платформи IntelliJ. Незважаючи на великий набір функцій, Rider - швидка IDE [14].

3. Крос-платформному виконанні. Rider не тільки вміє запускати і налагоджувати різні додатки в різних операційних системах, а й саме середовище розробки працює на Windows, macOS і Linux [14].

На рисунку 4.6 зображено приклад роботи у середовищі Rider для аналізу коду.

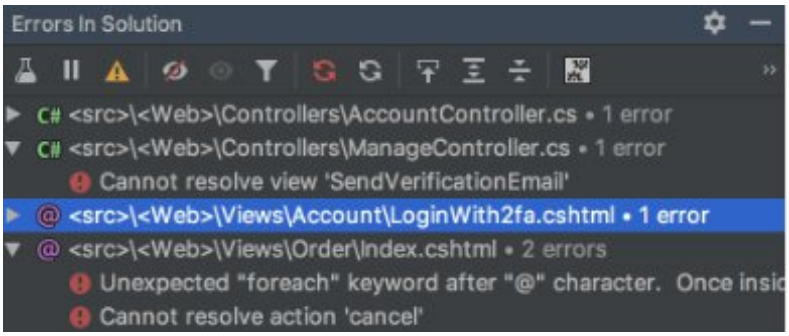

Рисунок 4.6 - Приклад аналізу коду в середовищі розробки Rider

На рисунку 4.7 зображено приклад редагування коду, використовуючи середовище розробки Rider.

Рисунок 4.6 - Приклад редагування коду в середовищі розробки Rider

Rider запозичив у ReSharper більше 60 рефакторингу і забезпечує понад 450 контекстних дій для різних цілей. Перебудовування спрощує перейменування та отримання методів, інтерфейсів та класів, переміщення та копіювання типів, використання альтернативного синтаксису та інших перетворень (Рисунок 4.7).

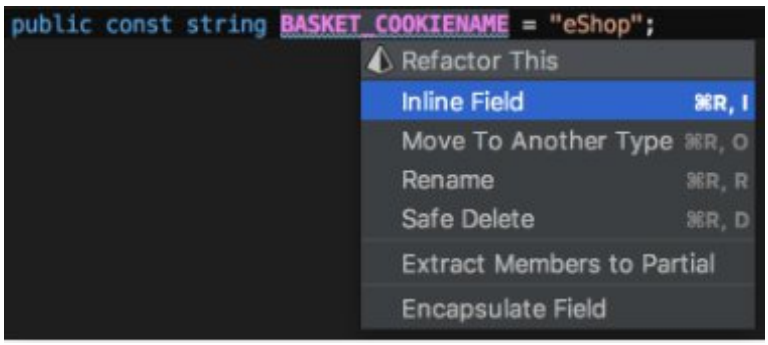

Рисунок 4.7 - Приклад рефакторингу коду в середовищі розробки Rider

Rider підтримує JavaScript, TypeScript, HTML, CSS та Sass. Спеціально для цих технологій IDE включає можливості рефакторингу, налагодження та тестування модулів від WebStorm (Рисунок 4.8).

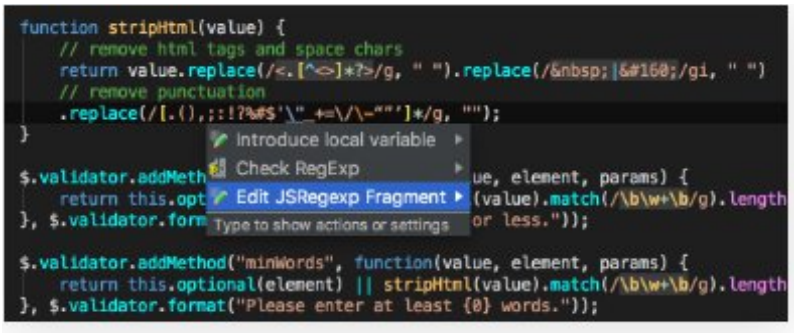

Рисунок 4.8 - Приклад використання фронтентд-технологій в середовищі розробки Rider

5. ОПИС РОЗРОБЛЕНОЇ СИСТЕМИ

Створене програмне рішення є веб-проектом.

Розробка системи поділялась на чотири етапи, а саме:

1. Розробка бізнес-логіки, яка містить всі необхідні об’єкти предметної області, геометричні розрахунки тощо.
2. Розробка контролеру, який пов’язує бізнес-логіку з графічним інтерфейсом.
3. Розробка графічного інтерфейсу користувача.
4. Налагодження системи та виявлення помилок.

5.1 Опис архітектури системи

Архітектура системи розділена на три компонента:

1. Бізнес-логіку(Модель)
2. Контролер
3. Графічний інтерфейс

Бізнес-логіка(Модель) - компонент архітектури системи, що містить опис предметної області та відповідальний за математичні розрахунки, реалізує всі необхідні алгоритми для побудови мінімальних поверхонь.

Контролер - компонент архітектури системи, відповідальний за обробку дій користувача, валідацію даних та зв’язок моделі з графічним інтерфейсом.

Графічний інтерфейс - компонент архітектури системи, основною задачою якого є візуалізація дискретних даних моделі, тобто графічне представлення розрахованих координат точок мінімальної поверхні та надання інструментів для більш детального дослідження.

Принцип роботи системи, використовуючи такий розподіл обов’язків, наступний: запит користувача направляється до Контролера, який відповідальний за

роботу роботу з бізнес-логікою(моделю) для виконання дій користувача та / або отримання результатів запиту. Контролер обирає необхідний графічний інтерфейс для відображення користувачу та надає необхідні дані Моделі.

На рисунку 5.1 зображено архітектуру системи та взаємодію її компонентів.

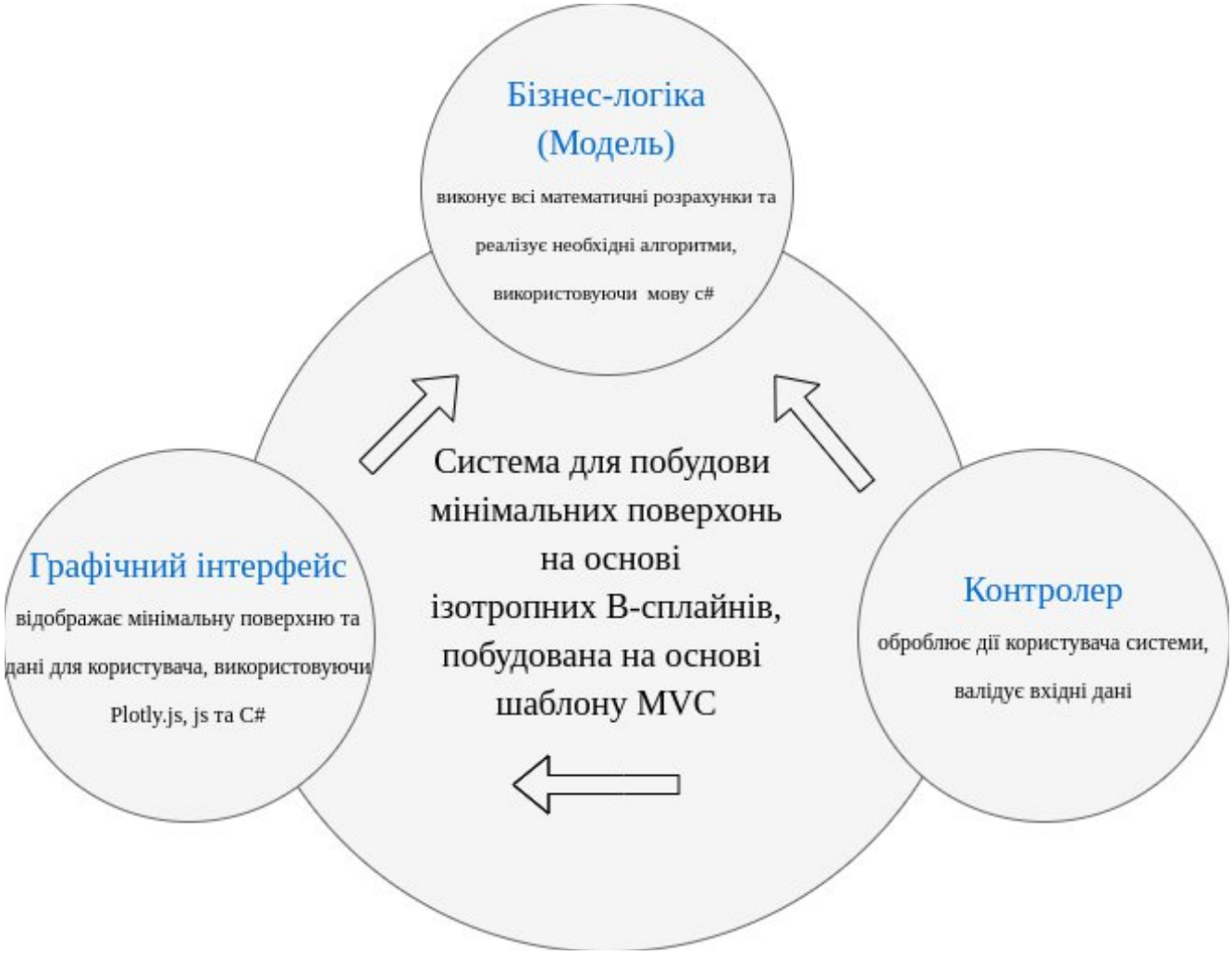

Рисунок 5.1 – Архітектура системи та взаємодія її компонентів

Вхідною інформацією для системи є файл з розширенням .json, який містить необхідні для побудови мінімальної поверхні початкові дані, вихідною інформацією - графічне відображення дискретних значень розрахованих координат точок поверхні та значення характеристичних коефіцієнтів для них(Рисунок 5.2).

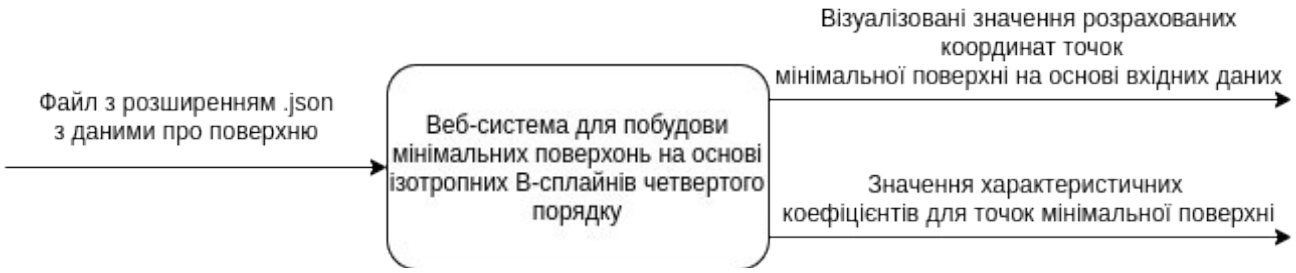

Рисунок 5.2 – Зображення вхідних та вихідних даних системи

5.2 Опис внутрішньої структури проекту

Так як програмне рішення спирається на ідеологію проектування MVC, яка дозволяє розділяти бізнес-логіку від графічногото представлення та змінювати певні частини системи без пливу на інші компоненти, відслідковувати помилки більш ефективно, структура проекту складається з трьох основних пакетів, а саме:

1. Controllers
2. Models
3. Views

та трьох додаткових:

1. Validation
2. ViewModels

Кожен з цих пакетів містить класи, які спроектовані для рішення конкретних задач і в злагодженій роботі один з одним в системі вирішують основну задачу - побудову мінімальної поверхні.

Внутрішня структура системи зображена на рисунку 5.3.

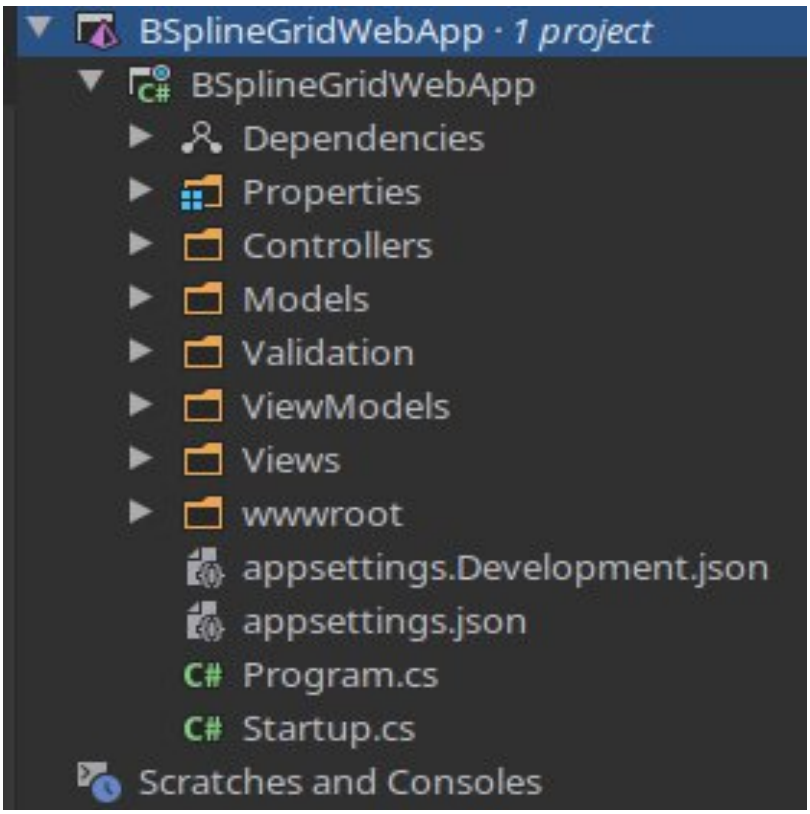

Рисунок 5.3 – Внутрішня структура створеного програмного рішення

Пакет Models містить необхідні класи для опису предметної області та реалізації необхідної бізнес-логіки, а саме: розрахунок уявних частин координат точок характеристичного чотирикутника, пошук дискретних значень базисних функцій, розрахунок координат точок мінімальної поверхні тощо. Вміст цього пакету є зображений на рисунку 5.4.

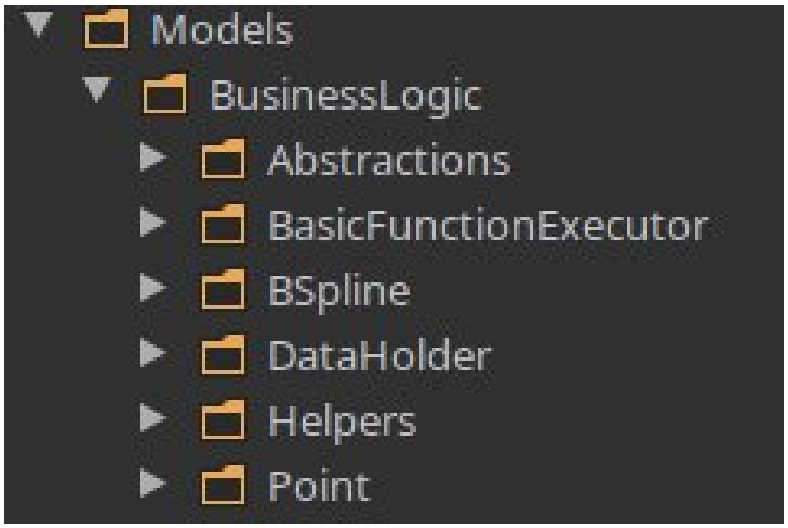

Рисунок 5.4 – Вміст пакету Models

Пакет Controllers містить клас GridController, що є посередником між бізнес-логікою системи та графічним інтерфейсом(представленням) користувача.

Пакет Views містить пакет Grid, який зберігає файл grid.cshtml, що є графічним інтерфейсом користувача.

Пакет Validation зберігає класи для валідації даних, які надходять до контролеру.

Пакет ViewModels зберігає моделі з даними, які користувач надсилає контролеру системи.

5.3 UML - діаграми основних класів

UML - діаграми основних класів бізнес-логіки наведено на рисунку 5.5 та 5.6.

На рисунку 5.7 зображено UML-діаграму контролеру системи.

На рисунку 5.8 зображено UML-діаграму допоміжних класів.

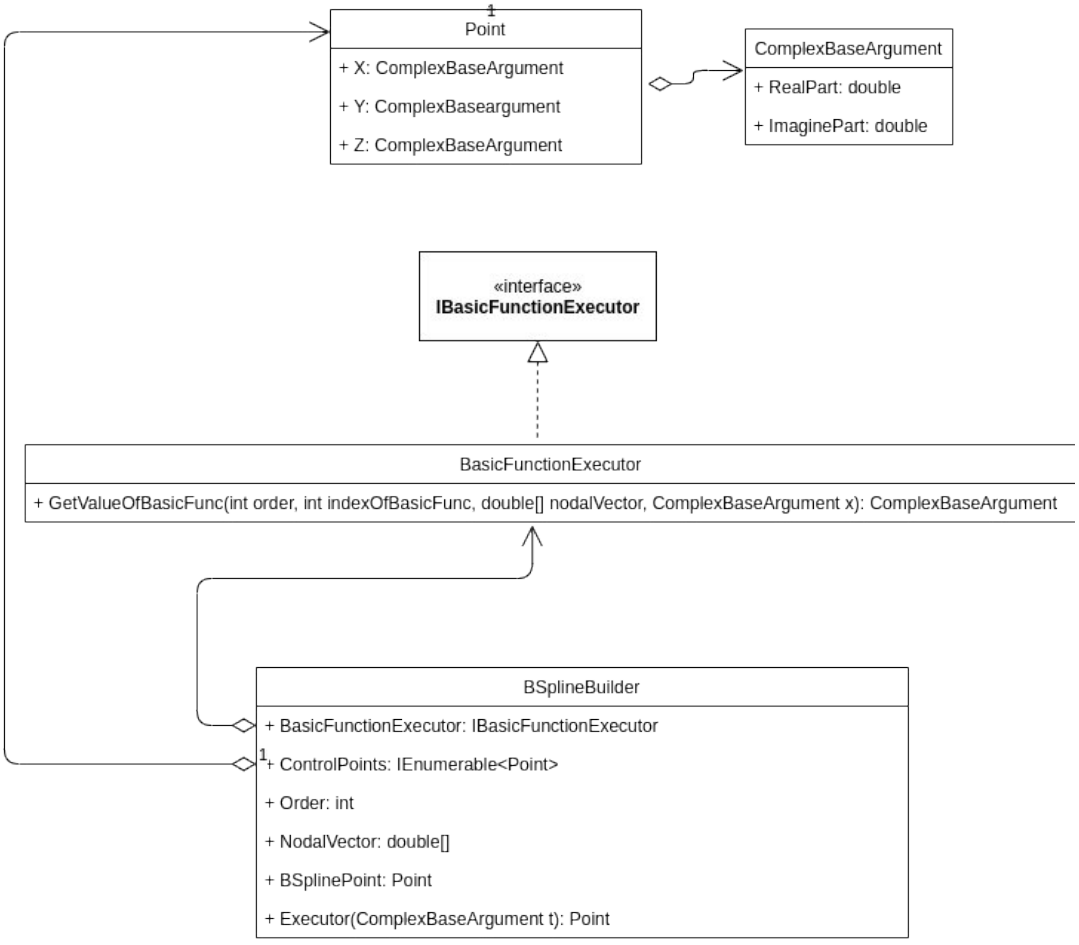

Рисунок 5.5 – UML - діграма основних класів бізнес-логіки системи

Рисунок 5.6 – UML - діграма основних класів бізнес-логіки системи

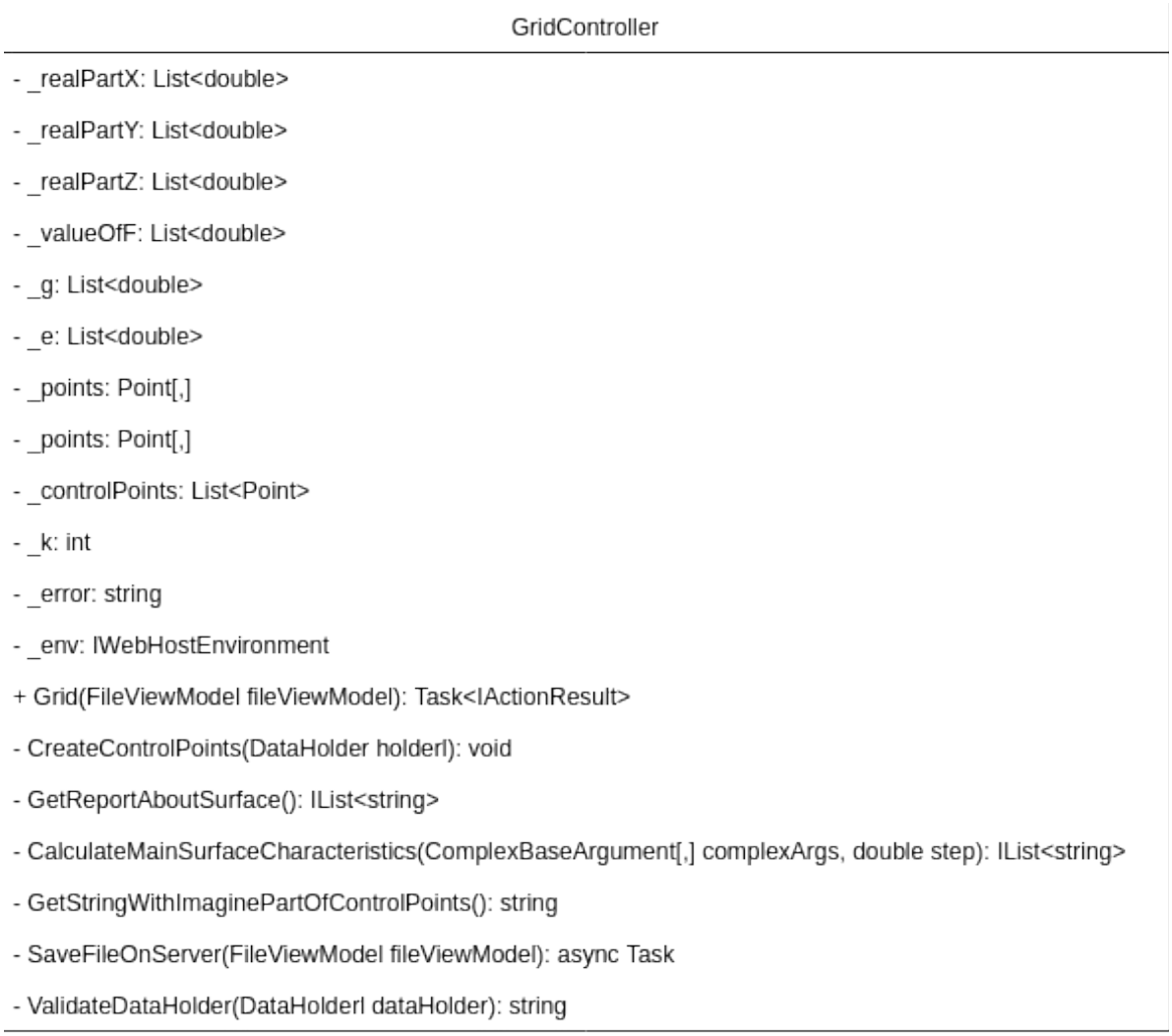

Рисунок 5.7 – UML - діграма класу-контролеру системи

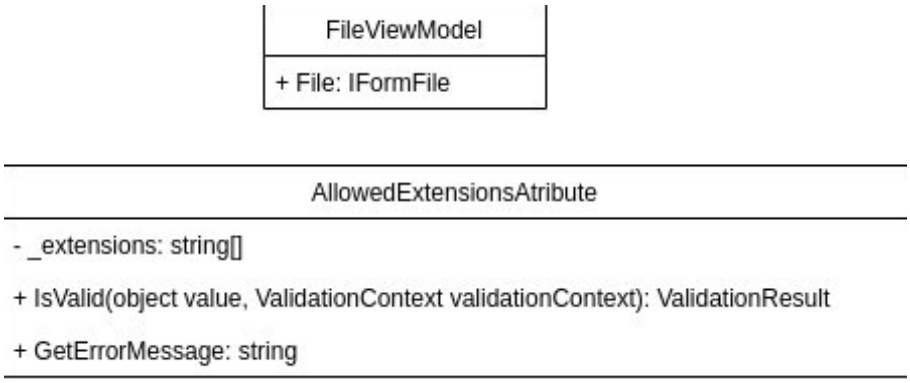

Рисунок 5.8 – UML - діграма допоміжних класів

5.4 Опис прогромної реалізації методу побудови мінімальної поверхні на основі ізотропних B-сплайнів четвертого порядку з квазіконформною заміною параметру

Головною метою створеного програмного забезпечення є вирішення проблеми побудови мінімальних поверхонь, які в якості напрямної кривої використовують ізотропні B-сплайни четвертого порядку з квазіконформною заміною параметру. Для цього було реалізовано відповідні класи та методи.

Одним з головних класів за допомогою якого система проводить розрахунки є клас BSplineBuilder.cs. Реалізовано класи зі статичними методами для таких цілей, як знаходження уявних частин комплексних координат точок характеристичного чотирикутника(ImaginePartDeterminant.cs), розрахунок частинних похідних за допомогою чисельних методів(ImaginePartDeterminant.cs), а також клас для збереження значення координат точок(Point.cs) тощо.

Алгоритм роботи системи є наступний:

1. Зчитування інформації про просторовий ізотропний B-сплайн четвертого порядку
2. Валідація отриманих даних
3. Розрахунок комплексних координат точок характеристичного чотирикутника
4. Формування вхідних аргументів для розрахунку координат поверхні
5. Розрахунок координат мінімальної поверхні
6. Виділення дійсної частини отриманих значень координат
7. Дослідження мінімальної поверхні, шляхом розрахунку коефіцієнтів E, G, F
8. Відображення поверхні та коефіцієнтів користувачу в графічному інтерфейсі

5.4.1 Реалізація алгоритму Кокса-де-Бура для знаходження значень базисних функцій

Важливим этапом для побудови просторових ізотпропних B-сплайнів є розрахунок значень базисних функцій.

Метод, що реалізує це завдання, має наступну сигнатуру (Рисунок 5.9):

```
public ComplexBaseArgument GetValueOfBasicFunc (int order, int indexOfBasicFunction, double[] nodalVector,
    ComplexBaseArgument x)
```

Рисунок 5.9 – Сигнатура методу для розрахунку значень базисних функцій

order - степінь B-spline’у.

indexOfBasicFunction - індекс базисної функції B-spline’у.

nodalVector - масив, що зберігає значення вузлового вектору.

x - комплексний аргумент, для якого розраховується значення базисної функції.

Метод GetValueOfBasicFunc має рекурсивну реалізацію.

Умовою виходу з рекурсії є рівність нулю степення B-spline’у(Рисунок 5.10)

```
if (order == 0)
{
    if (x >= nodalVector.ElementAt(indexOfBasicFunction) &&
        x < nodalVector.ElementAt(indexOfBasicFunction + 1))
    {
        result = new ComplexBaseArgument(1,0);
    }

    else
    {
        result =  new ComplexBaseArgument(0,0);
    }

}
```

Рисунок 5.10 – Умова виходу з рекурії в методі пошуку значень базисних функцій

Якщо степінь B-spline’у дорівнює нулю, тоді відбувається перевірка, чи лежить дісна частина комплексного аргументу в межах значень елементів вузлового вектору, індекс яких знаходиться в межах [indexOfBasicFunction; indexOfBasicFunction + 1).

Якщо ця умова виконується, тоді повертається значення функії у комплексному вигляді з дійсною частиною рівною одиниці, в іншому випадку повертається комплексне число з дійсною частиною рівною нулю.

Якщо степінь B-spline’у ще не рівний нулю, тоді виконується рекурсивна частина методу, що задається формулою 3.10.

Оскільки у визначені алгоритму Кокса де Бура 0/0 = 0, спочатку розраховуються окремо чисельники та знаменники даного виразу.

Для початку розрахуємо знаменники;

Для першого знаменника знайдемо різницю значень елементів, що знаходяться під індексами indexOfBasicFunction + order та indexOfBasicFunction.

Для другого знаменника знайдемо різницю значень елементів, що знаходяться під індексами indexOfBasicFunction + order + 1 та indexOfBasicFunction + 1.

```
firstDenominator = nodalVector[indexOfBasicFunction + order] - nodalVector[indexOfBasicFunction];
secondDenominator = nodalVector[indexOfBasicFunction + order + 1] -
                    nodalVector[indexOfBasicFunction + 1];
```

Рисунок 5.11 – Розрахунок значень знаменників у формулі 3.10

Далі йде перевірка рівності знаменників нулю.

Для першого дробу, якщо знаменник рівний нулю, тоді значення дробу прирівнюється до значення комплексного числа, з дійсною та уявною частинами рівними нулю, в іншому випадку розраховується значення дробу. Для цього знаходиться різниця між значенням комлексного аргументу та значенням елементу вузлового вектору під індексом indexOfBasicFunction, яка ділиться на значення знаменника, який було знайдено на попередньому кроці (Рисунок 5.12).

```
if (Math.Abs(firstDenominator) < 0.0000001)
{
    firstMultiplier = new ComplexBaseArgument(0,0);
}
else
{
    firstMultiplier = (x - nodalVector[indexOfBasicFunction]) / firstDenominator;
}
```

Рисунок 5.12 – Розрахунок значення першого множника у формулі 3.10

Аналогічно до другого дробу (Рисунок 5.13).

```
if (Math.Abs(secondDenominator) < 0.0000001)
{
    secondMultiplier = new ComplexBaseArgument(0,0);
}
else
{
    secondMultiplier = (nodalVector[indexOfBasicFunction + order + 1] - x) / secondDenominator;
}
```

Рисунок 5.13 – Розрахунок значення першого множника у формулі 3.10

Далі безпосередньо знаходиться значення базисної функції, що відповідає початковим вхідним параметрам.

Для цього значення першого дробу множимо на значення фунції GetValueOfBasicFunc(order - 1, indexOfBasicFunction, nodalVector, x), та значення другого дробу множимо на значення функції GetValueOfBasicFunc(order - 1, indexOfBasicFunction + 1, nodalVector, x) та знаходимо їх суму.

Блох-схема методу GetValueOfBasicFunc, що реалізує алгоритм Кокса-де Бура зображено на рисунку 5.14.

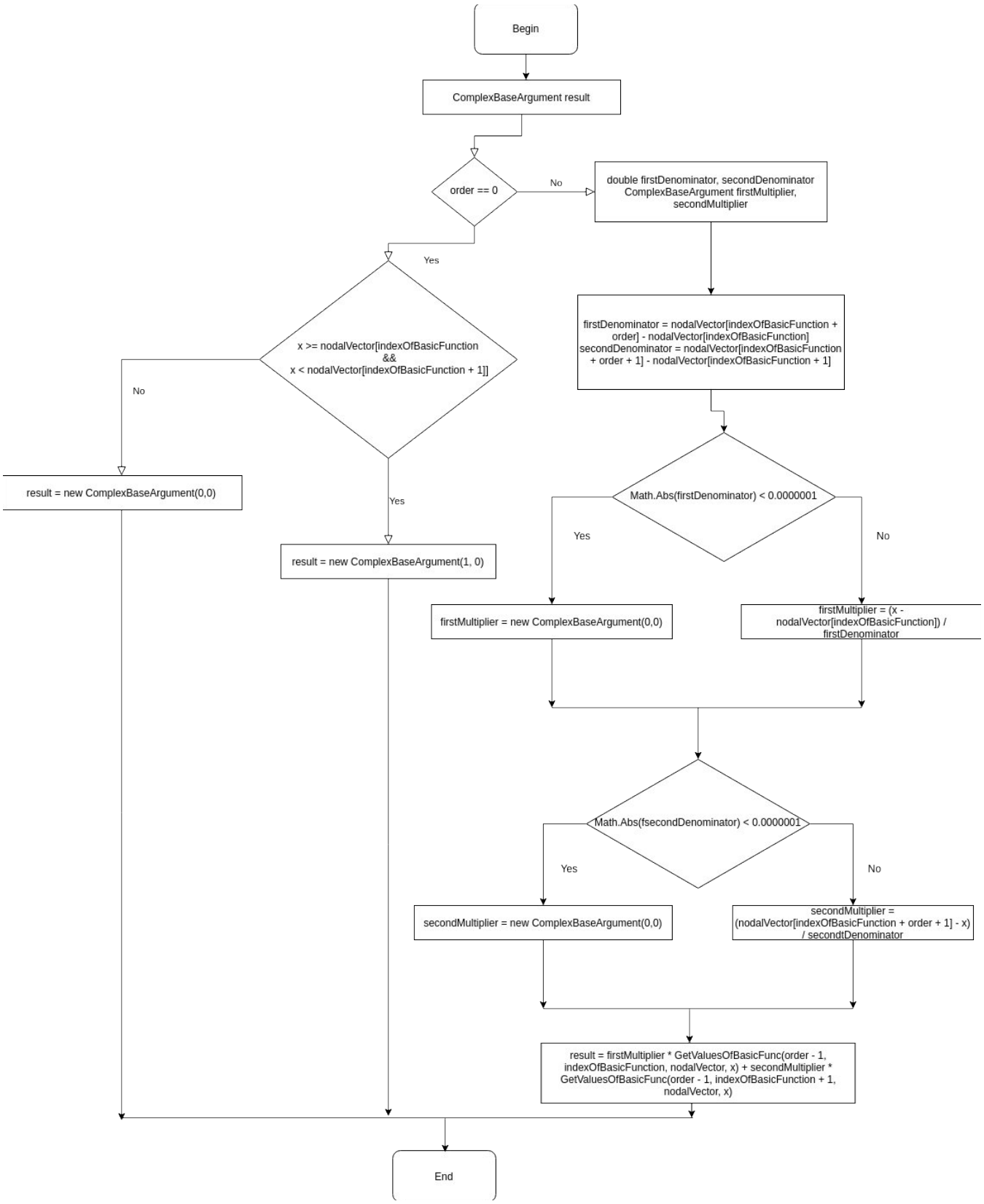

Рисунок 5.14 - Блок-схема методу GetValueOfBasicFunc, що реалізує алгоритм Кокса-де Бура

5.4.2 Реалізація методу розрахунку комплесної просторової координидинати для мінімальної поверхні

Для знаходження координат точок поверхні створено клас BSplineBuilder.cs, що містить метод Execute(ComplexBaseArgument t), який в якості параметру отримує комплексне число. UML - діаграму класу відображено на оисунку 5.5. Результатом роботи методу є екземпляр класу Point.cs, що містить комплесні значення координат x, y, z точки, що належить поверхні. Тіло методу для початку ініціалізує значення BSplinePoint екземпляром класу Point.cs, який в свою чергу ініціалізується комплесними числами зі значенням дійсної та уявної частини рівними нулю. Наступним етапом є ініціалізація лічильника індексу базисної функції indexOfBasicFunction значеням, рівним нулю. Далі в циклі foreach перебираємо значення контрольних точок(точки характеристичного чотирикутника), знаходимо значення базисної функції для відповідного індексу базисної функції(виклик методу GetValueOfBasicFunc класу BasicFunctionExecutor), вхідного параметру t, степення B-сплайну, вектору вузлових значень. Після знаходження значення базисної функції до існуючих значень координат об’єкту BSplinePoint додаються відповідні значення координат контрольної точки, помножене на знайдене значення базисної функції. В кінці кожної ітерації циклу значення лічильника indexOfBasicFunction збільшується на одиницю. Реалізація методу наведена на рисунку 5.15.

```
public global::BSplineGridWebApp.Models.BusinessLogic.Point.Point Execute(ComplexBaseArgument t)
{
    BSplinePoint = new global::BSplineGridWebApp.Models.BusinessLogic.Point.Point( x: new ComplexBaseArgument( realPart: 0, imaginePart: 0),
        y: new ComplexBaseArgument( realPart: 0, imaginePart: 0),
        z: new ComplexBaseArgument( realPart: 0, imaginePart: 0));
    int indexOfBasicFunction = 0;
    foreach (var controlPoint in ControlPoints)
    {
        ComplexBaseArgument valueOfBasicFunc =
            BasicFunctionExecutor.GetValueOfBasicFunc(Order, indexOfBasicFunction, NodalVector, t);
        BSplinePoint.X += controlPoint.X * valueOfBasicFunc;
        BSplinePoint.Y += controlPoint.Y * valueOfBasicFunc;
        BSplinePoint.Z += controlPoint.Z * valueOfBasicFunc;
        indexOfBasicFunction++;
    }

    return BSplinePoint;
}
```

Рисунок 5.15 – Реалізація методу Execute класу BSplineBuilder

Так як ми маємо масив комплексних параметрів complexArgs, то для кожного елементу цього масиву викликається метод Execute класу BSplineBuilder.

Блок-схема алгоритму, що реалізує метод Execute зображено на рисунку 5.16.

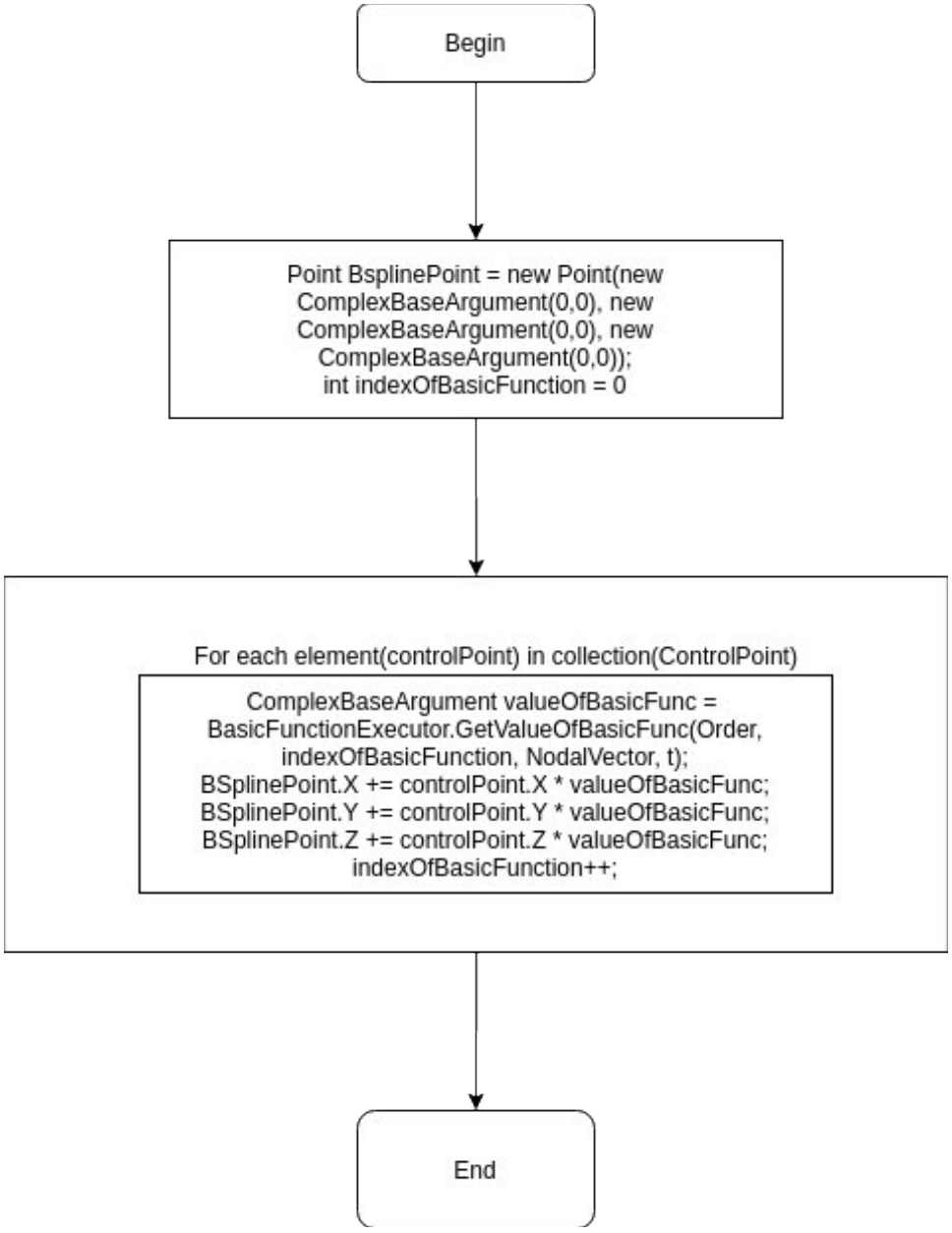

Рисунок 5.16 Блок-схема алгоритму, який реалізує метод Execute класу BSplineBuilder

5.4.3 Реалізація класу для роботи з алгеброю комплесних чисел

Так як для побудови кривих нульової довжини необхідно використовувати арифтметичні операції для роботи з комплесними значеннями було спроектовано

клас ComplexBaseArgument, який зберігає дійсну та уявну частини комплесного числа та має всі необхідні методи для проведення арифметичних та логічних операцій над комплесним типом даних, а саме:

- Операції порівняння(<, >, <=, >=) комплесного числа(дійсної частини) з числами з плаваючою точкою
- Арифметичні операції, а саме: додавання, віднімання, ділення, множення комлесних чисел, комплесного числа і числа з плаваючою точкою
- Операція піднесення комлесного числа до цілого степеня
- Операція унарного мінуса для комлесного числа

Всі методи реалізовані за допомогою такої технології мови програмування C# як перевантаження операцій.

Приклад реалізації таких операцій наведено нижче(Рисунок 5.17).

```
public static ComplexBaseArgument operator +(ComplexBaseArgument first, ComplexBaseArgument second)
{
    return new ComplexBaseArgument(realPart: first.RealPart + second.RealPart,
        imaginePart: first.ImaginePart + second.ImaginePart);
}

public static ComplexBaseArgument operator -(ComplexBaseArgument first, ComplexBaseArgument second)
{
    return new ComplexBaseArgument(realPart: first.RealPart - second.RealPart,
        imaginePart: first.ImaginePart - second.ImaginePart);
}

public static ComplexBaseArgument operator -(ComplexBaseArgument complexArg)
{
    return new ComplexBaseArgument( realPart: 0.0-complexArg.RealPart,  imaginePart: 0.0-complexArg.ImaginePart);
}
```

Рисунок 5.17 – Приклад перевантажених операцій для роботи з комплексними числами

5.5 Графічний інтерфейс користувача

Для побудови графічного інтерфейсу в ASP.NET Core існує представлення. Представлення - це файли з розширенням .chtml, що містять в собі код користувацького інтерфейсу в основному на мові розмітці HTML5. Хоч представлення містить в собі html-розмітку, воно не є html-сторінкою. Стандартне представлення дуже схоже на html, але воно також містить вставки коду на мові

програмування C#, перед якими йде символ @. Цей знак використовує Razor для переходу до коду C#.

Створення інтерфейсу відбувалося в два етапи:

1. Створення розмітки сторінки, визначення необхідних елементів інтерфейсу, що відповідатимуть потребам користувача, визначення каскадних стилів
2. Зв’язок інтерфейсу з контролером для їх комунікації, обміну даними, необхідними для бізнес-логіки

Початковий інтерфейс системи наведено нижче (Рисунок 5.18).

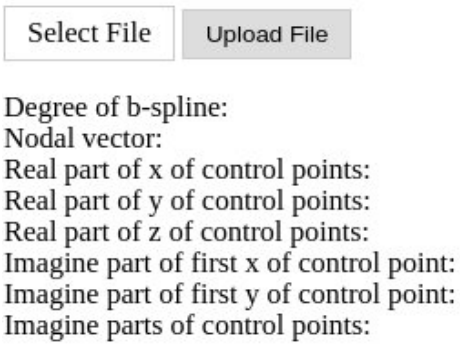

Рисунок 5.18 – Початковий графічний інтерфейс користувача

Інтерфейс, зображений на (Рисунок 5.18) має наступні елементи:

- Кнопка керування Select File, яка дозволяє обрати файл з розширенням .json, який містить необхідні дані на основі яких необхідно побудувати поверхню
- Кнопка керування Upload File, яка надсилає вибраний користувачем файл на сервер і далі система починає працювати із завантаженим файлом
- Відображення даних на основі яких буде відображатися поверхня, а саме: степінь B-сплайну, вектор вузлових значень, дійсні частини координат x, y, z

точок характеристичного чотирикутника, уявні частини перших координат x, y, а також уявні частини координат z точок характеристичного чотирикутника

Так як файл з даними не загружений, поверхня не відображається.

Після того як користувач загрузив файл з необхідними для правильної роботи системи даними графічний інтерфейс має вигляд (Рисунок 5.19)

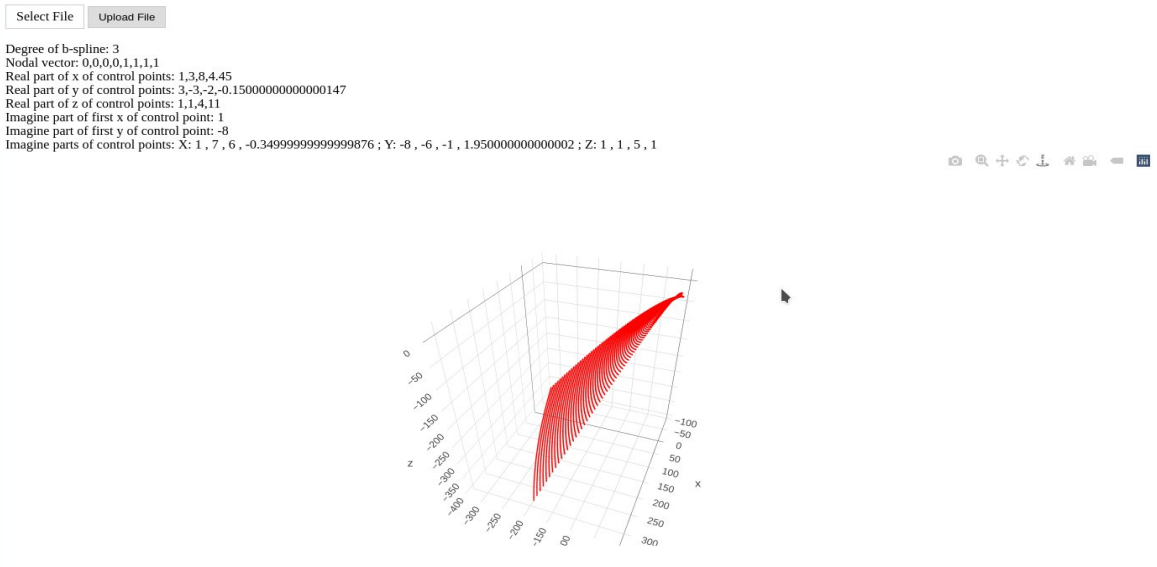

Рисунок 5.19 – Графічний інтерфейс системи після опрацювання файлу з даними, які завантажив користувач

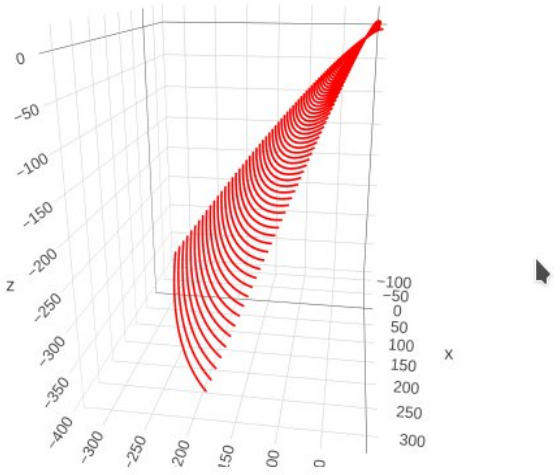

Рисунок 5.20 – Мінімальна поверхня на основі ізотропного B-сплайну четвертого порядку з квазіконформною заміною параметру

Окрім самої поверхні, елементом інтерфейсу є блок в якому демонструються значення характеристичних параметрів мінімальної поверхні (Рисунок 5.21).

Orthogonal and isothermal characteristics

X:	-0.70666	Y:	1.34145	Z:	0.78474	F:	0.01479	G: 1049.83972	E: 1049.81387
X:	-0.84283	Y:	1.25460	Z:	0.75883	F:	0.01498	G: 1077.83819	E: 1077.81164
X:	-0.97979	Y:	1.16526	Z:	0.73141	F:	0.01517	G: 1106.49976	E: 1106.47249
X:	-1.11755	Y:	1.07342	Z:	0.70244	F:	0.01536	G: 1135.83593	E: 1135.80795
X:	-1.25610	Y:	0.97906	Z:	0.67191	F:	0.01555	G: 1165.85837	E: 1165.82966
X:	-1.39544	Y:	0.88218	Z:	0.63977	F:	0.01574	G: 1196.57882	E: 1196.54938
X:	-1.53556	Y:	0.78275	Z:	0.60600	F:	0.01593	G: 1228.00917	E: 1227.97899
X:	-1.67646	Y:	0.68077	Z:	0.57057	F:	0.01612	G: 1260.16141	E: 1260.13047
X:	-1.81814	Y:	0.57623	Z:	0.53344	F:	0.01631	G: 1293.04765	E: 1293.01596
X:	-1.96058	Y:	0.46911	Z:	0.49459	F:	0.01650	G: 1326.68012	E: 1326.64767

Рисунок 5.21 - Фрагмент таблиці на якій зображено координати точок мінімальної поверхні та значення коефіцієнтів F, G, E у цих точках

Бібліотека plotly.js надає гнучкі можливості для роботи з трьохвимірними об’єктами, а саме:

1. Збереження побудованої поверхні у форматі .png (Рисунок 5.22)
2. Збільшення конкретної ділянки побудованої поверхні (Рисунок 5.23)
3. Обертання поверхні для перегдяду її під різними кутами для детальнішого аналізу (Рисунок 5.24)
4. Перегляд значень координат точки, на яку було наведено курсор миші (Рисунок 5.25)
5. Переміщення поверхні вздовж виділеної під поверхню області
6. Орбітальне обертання поверхні
7. Поворотне обертання поверхні
8. Повернення до початкового положення поверхні
9. Повернення до останього збереженого положення поверхні

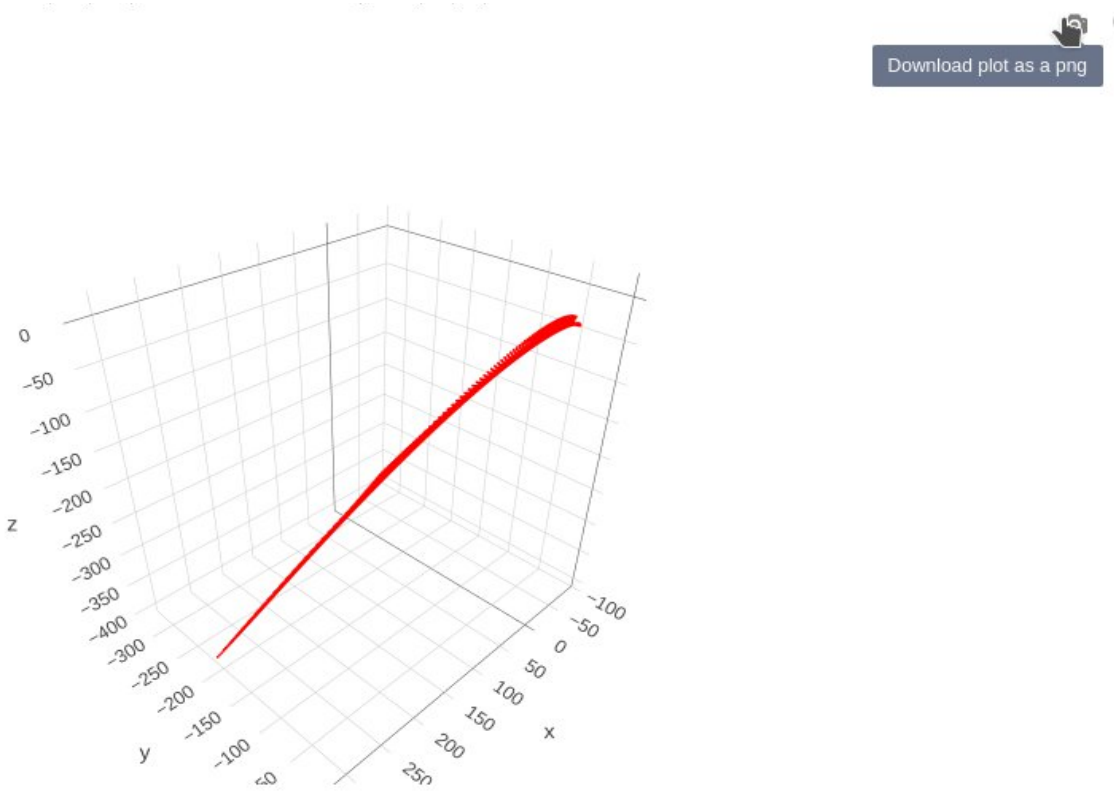

Рисунок 5.22 - Збереження побудованої мінімальної поверхі

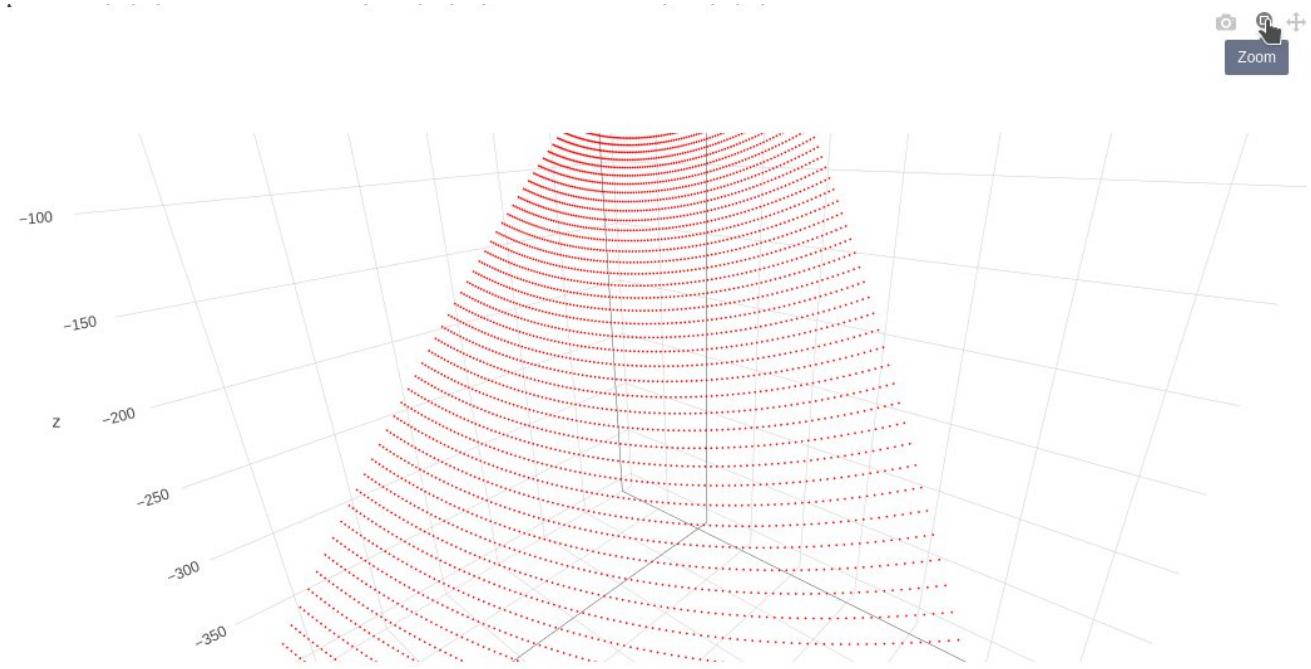

Рисунок 5.23 - Збільшення ділянки побудованої поверхні

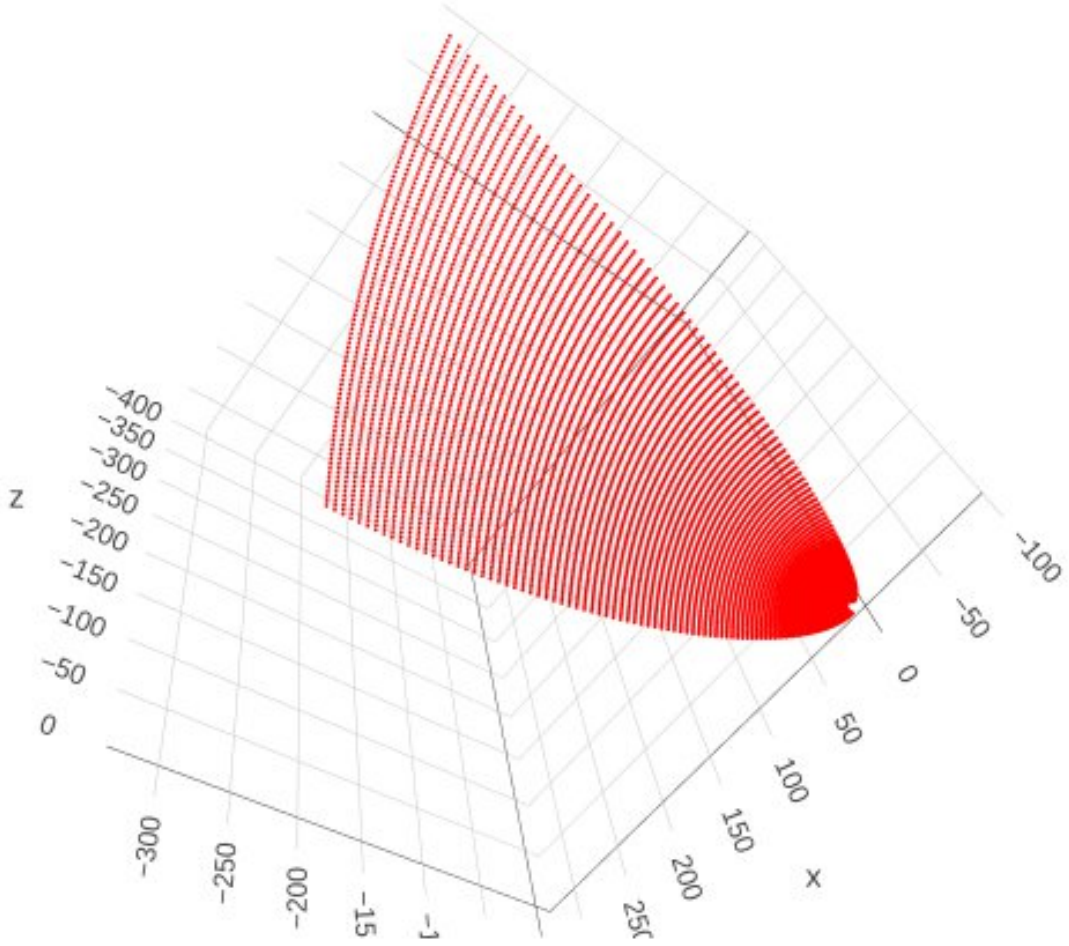

Рисунок 5.24 - Обертання поверхні для перегляду її під різними кутами

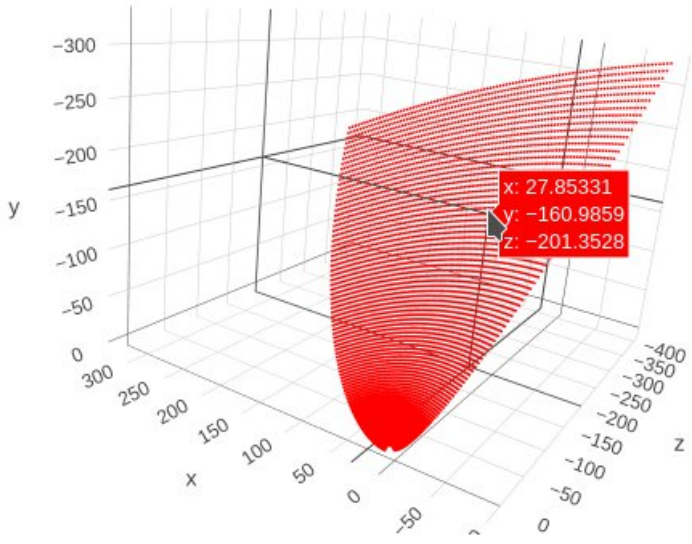

Рисунок 5.25 - Перегляд значення координати точки на яку наведено курсор миші

Також було реалізовано механізм повідомлення користувача системи про помилки, які пов’язані з коректністю наданих ним даних, за допомогою такої функції мови програмування JS як alert().

На рис. 5.25 зображено повідомлення про помилку, коли користувач обрав файл не з розширенням .json.

Рисунок 5.25 - Повідомлення, сгенероване системої, про некоректність розширення завантаженого користувачем файлу

6. РОБОТА КОРИСТУВАЧА З ПРОГРАМНИМ ПРОДУКТОМ

Для гарантування правильної, безвідмовної роботи розробленої системи для побудови мінімальних поверхнонь на основі ізотропних B-сплайнів четвертого порядку з квазіконформною заміною параметру необхідно дотримуватися необхідних інструкцій щодо умов запуску системи, встановлення додаткового програмного забезпечення та рекомендацій при використані системи. Якщо користувач розгортає програмне забезпечення в себе на комп’ютері, тобто його обчислювальна машина виступає в якості сервера, на якому запускається веб-система, то є необхідність встановлення додаткового програмного забезпечення. У іншому випадку для роботи з системою нема необхідності встановлення додаткового програмного забезпечення, окрім того, що користувач повинен мати веб-браузер, який підтримує JS.

6.1 Системні вимоги

Для розгортання системи необхідно мати одну з операційних систем: Windows, Linux, maxOs та необхідний обсяг вільної пам’яті на жорсткому диску, а саме не менше 6.1 MiB для самої системи. Необхідно переконатися, що на обчислювальній машині встановлено .Net Core SDK 3.1 та .Net Core Runtime 3.1.

6.2 Рекомендації щодо використання

Для першого запуску системи необхідно:

1. Мати файл BSplineMinimalSurfaceWebApp.zip

2. Далі необхідно розархівувати його
3. Перейти у папку за наступною адресою, а саме: BSplineMinimalSurfaceWebApp/BSplineGridWebApp/BSplineGridWebApp/
4. Відкрити вікно терміналу цій папці та ввести команду dotnet run
5. Відкрити браузер та перейти за адресою http://localhost:5000

Для початку система повідомить про необхідність обрати та завантажити файл з необхідними для побудови поверхні даними у форматі .json (Рисунок 6.1).

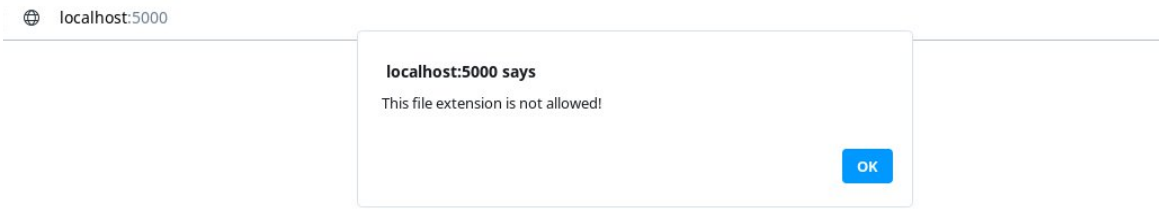

Рисунок 6.1 - Початкове вікно веб-системи після запуску

Далі необхідно обрати файл у форматі .json, який відповідає наступним вимогам:

1. Містить поле Degree зі встановленим значенням 3
2. Містить поле RealPartXOfControlPoints
3. Містить поле RealPartYOfControlPoints
4. Містить поле RealPartZOfControlPoints
5. Містить поле ImaginePartZOfControlPoints
6. Містить поле ImaginePartOfFirstX
7. Містить поле ImaginePartOfFirstY
8. Містить поле NodalVector

Приклад заповненого файлу зображено на рисунку 6.2.

```
{
  "Degree" : 3,
  "RealPartXOfControlPoints" : [1, 3, 8, 0],
  "RealPartYOfControlPoints" : [3, -3, -2, 0],
  "RealPartZOfControlPoints" : [1, 1, 4, 0],
  "ImaginePartZOfControlPoints" : [1, 1, 5, 0],
  "ImaginePartOfFirstX" : 1,
  "ImaginePartOfFirstY" : -8,
  "NodalVector" : [0, 0, 0, 0, 1, 1, 1, 1]
}
```


Рисунок 6.2 - Приклад заповнення файлу з розширенням .json для побудови мінімальної поверхні

Після створення та заповнення файлу його необхідно завантажити в систему. Для цього необхідно спочатку натиснути на кнопку Select File та знайти щойно створений файл (Рисунок 6.3).

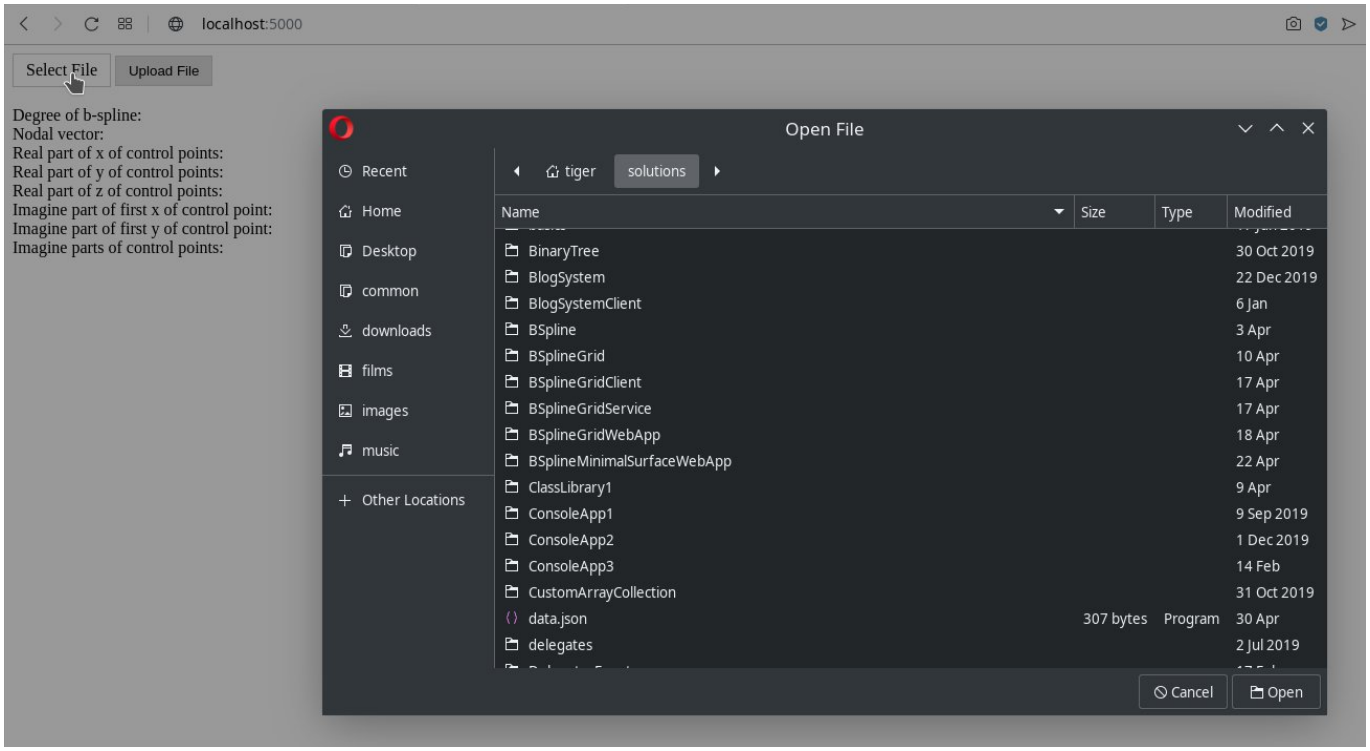

Рисунок 6.3 - Вибір файлу з даними для побудови мінімальної поверхні

Наступним кроком після вибору файлу його необхідно надіслати на сервер, натиснувши кнопку Upload File, яка в свою чергу після завантаження файлу дасть системі сигнал про початок обробки файлу (Рисунок 6.4).

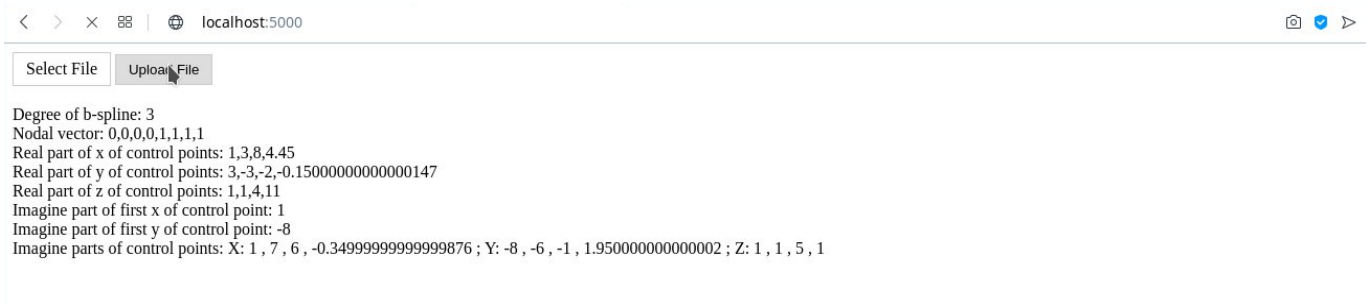

Рисунок 6.4 - Надсилання системі файлу з даними та початок обробки системою файлу

Якщо користувач ввів всі дані коректно веб-система відобразить побудовану поверхню (Рисунок 6.5) та таблицю зі значеннями характеристичних коефіцієнтів мінімальної поверхні. (Рисунок 6.6).

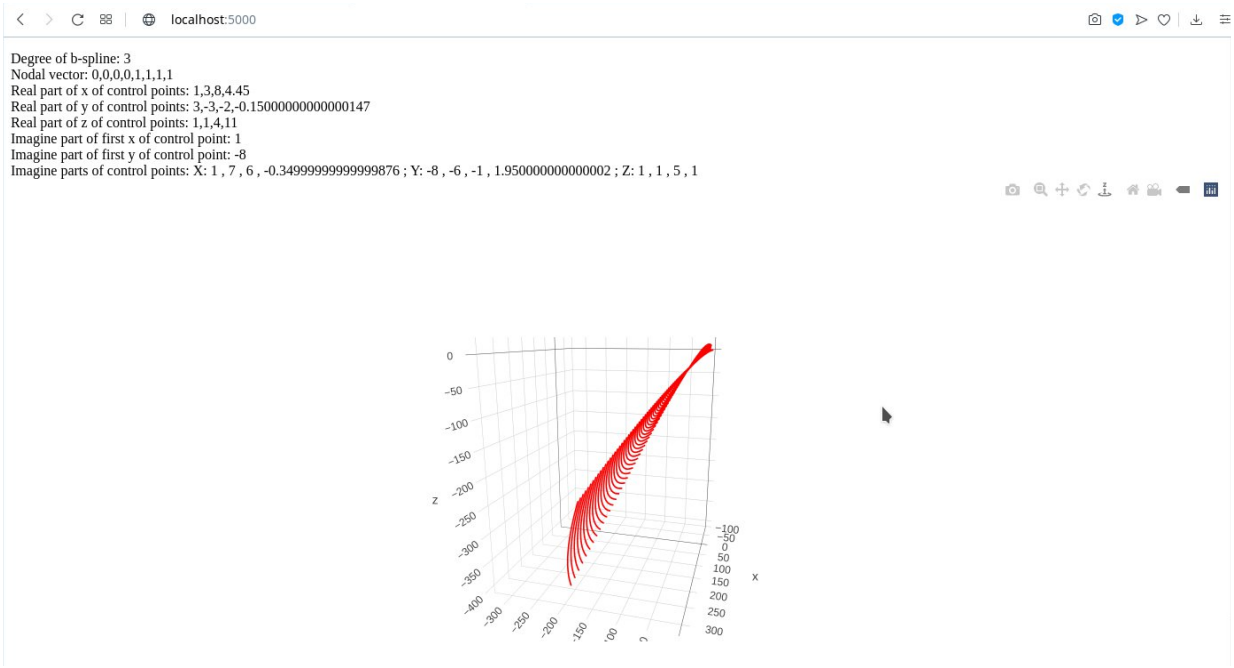

Рисунок 6.5 - Побудована поверхня на основі даних файлу, зображених на рис. 6.2

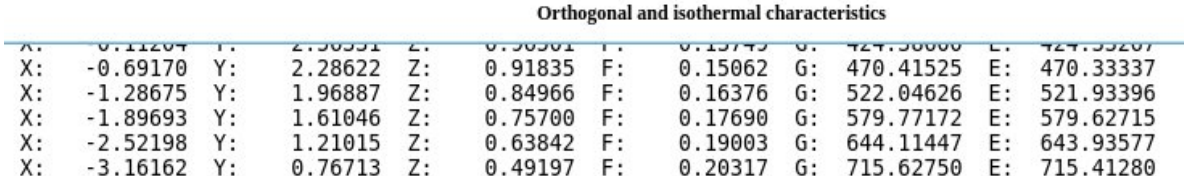

Рисунок 6.6 - Фрагмент таблиці зі значеннями характеристичних коефіцієнтів мінімальної поверхні, побудованих на основі даних, зображених на рис. 6.2

Після того як система відобразила мінімальну поверхню на основі вхідних даних, користувач може робити наступні кроки, саме:

- Зберегти поверхню у вигляді зображення з розширенням .png.
- Обертати поверхню, переглядаючи її під різними кутами.
- Збільшувати у розмірі певні ділянки поверхні.
- Повертатися до попереднього положення поверихні.
- Повертатися до початкового положення поверхі.

Переміщувати поверхню вздовж робочої області, що була виділена під поверхню

Для зупинки роботи серверу необхідно перейти за шляхом BSplineMinimalSurfaceWebApp/BSplineGridWebApp/BSplineGridWebApp/ та в цій папці відкрити вікно терміналу та натиснути комбінацію клаівш Ctrl + C та закрити сторінку у браузері за адресою http://localhost:5000.

6.3 Діаграма прецедентів

Діаграма прецедентів - це представлення взаємодії користувача з системою, яка показує взаємозв'язок між користувачем та різними випадками використання, в яких користувач бере участь. Діаграма випадків використання може ідентифікувати різні типи користувачів системи та різні випадки використання, а також часто супроводжуватися іншими типами діаграм. Випадки використання представлені або колами, або еліпсами.

Хоча сам випадок використання може детально описувати кожну можливість, діаграма прецедентів може допомогти забезпечити вигляд вищого рівня системи. Вони забезпечують спрощене та графічне зображення того, що система насправді повинна робити.

На рисунку 6.7 зображено діаграму прецедентів для розробленої системи.

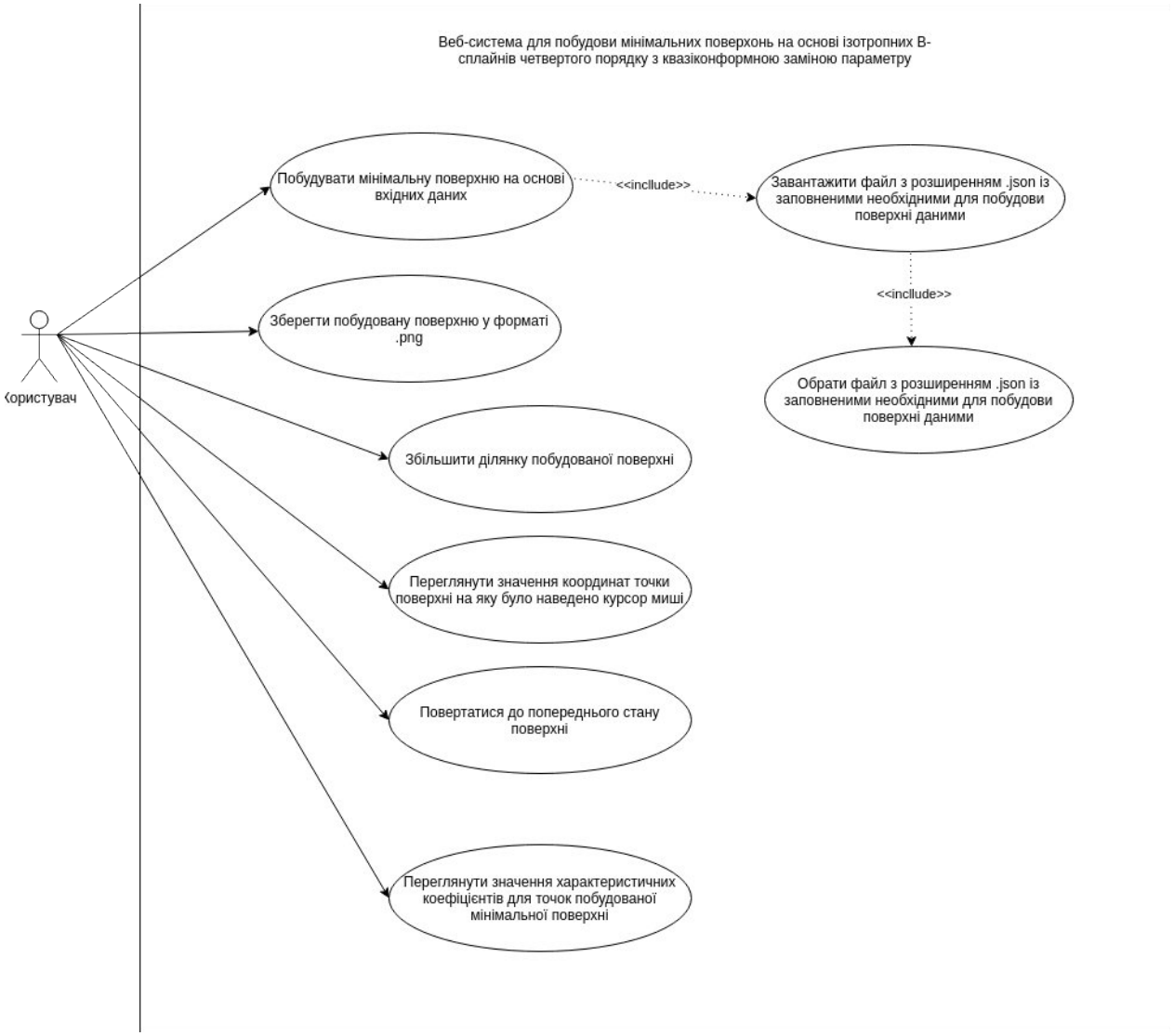

Рисунок 6.7 - Діаграма прцедентів розробленої системи

ВИСНОВКИ

Дипломну роботу присвячено актуальній темі побуди мінімальних поверхонь на основі B-сплайнів з квазіконформною заміною параметру. При виконанні поставлених задач отримано наступні результати:

1. Опрацьовано теоретичний матеріал, що пов’язаний з темою мінімальних поверхонь, кривих Безьє, ізотропних кривих Безьє, B-сплайнів, ізотропних B-сплайнів, ізотропних плоских сіток;
2. В якості методу побудови мінімальних поверхонь було обрано метод деформації плоских кривих, в якості кривої обрано B-сплайн третього порядку, в кривій проводиться квазіконформна заміна параметру;
3. Обрано засоби для програмної реалізації поставлених задач, а саме платформа ASP.NET Core MVC 3.1, об’єктно-орієнтована мова програмування C#, мова розмітки HTML5, таблиця каскадних стилів CSS3, мова програмувння JS, а також бібліотека plotly.js;
4. Створено структуру проекту, що відповідає технології MVC, тобто бізнес-логіка, представлення та обробник дій користувача розміщені в окрепих папках;
5. Розроблено алгоритмічну базу для розрахунку координат мінімальної поверхні, використовуючи B-сплайни четвертого порядку з квазіконформною заміною параметру;
6. Розроблено програмний продукт для побудови мінімальних поверхонь на основі B-сплайнів четвертого порядку з квазіконформною заміною параметру.

Програмне забезпечення створено за допомогою платформи ASP.NET Core MVC 3.1 та об’єктно-орієнтованої мови програмування C#.

В ході дипломної роботи проведено дослідження методу побудови мінімальних поверхонь на основі B-сплайнів четвертого порядку з квазіконформною заміною параметру, який базується на деформації плоских кривих, а також розроблено

програмний продукт, основними задачами якого є побудова мінімальних поверхнонь на основі B-сплайнів четвертого порядку з квазіконформною заміною параметру, використовуючи вхідну інформацію користувача про мінімальну поверхню, розрахунок характеристичних коефіцієнтів для точок мінімальної поверхні.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Д. Роджерс, Дж. Адамс Математические основы машинной графики / Д. Роджерс, Дж. Адамс //М.: Мир, 2001. - 221-222 с.
2. Д. Роджерс, Дж. Адамс Математические основы машинной графики / Д. Роджерс, Дж. Адамс //М.: Мир, 2001. - 297 с.
3. Д. Роджерс, Дж. Адамс Математические основы машинной графики / Д. Роджерс, Дж. Адамс //М.: Мир, 2001. - 310 с.
4. Д. Роджерс, Дж. Адамс Математические основы машинной графики / Д. Роджерс, Дж. Адамс //М.: Мир, 2001. - 311 с.
5. Д. Роджерс, Дж. Адамс Математические основы машинной графики / Д. Роджерс, Дж. Адамс //М.: Мир, 2001. - 317 с.
6. Д. Роджерс, Дж. Адамс Математические основы машинной графики / Д. Роджерс, Дж. Адамс //М.: Мир, 2001. - 314 с.
7. Д. Роджерс, Дж. Адамс Математические основы машинной графики / Д. Роджерс, Дж. Адамс //М.: Мир, 2001. - 316 с.
8. Н.М. Аушева, А.Г. Гурін. Моделювання плоских сіток на основі ізотропних B-сплайнів / Н.М. Аушева, А.Г. Гурін/ Вестник ХНТУ №3(54), 2015 г., с. 528-532
9. Н.М. Аушева. Розробка узагальненого підходу щодо формувань кривих та поверхонь дійсного простору на основі ізотропних характеристик/Н.М.Аушева/Технологический аудит и резервы производСтва — № 3/1(17), 2014, ст. 17-20
10. Ausheva N, Olevskyi V., Olevska Y. Modeling of Minimal Surface Based on an Isotropic Bezier Curve of Fifth Order. *Journal of Geometry and Symmetry in Physics (JGSP).* Bulgarian Academy of Sciences, 2019. Vol. 52. P. 1-15.
11. ASP NET MVC Pattern [Електронний ресурс] - https://dotnet.microsoft.com/apps/aspnet/mvc

12. Plotly Graphing Libraries [Електронний ресурс] - https://plotly.com/javascript/
13. JSON[Електронний ресурс] - https://www.json.org/json-en.html
14. Rider[Електронний ресурс] - https://www.jetbrains.com/ru-ru/rider/
15. Matlab [Електронний ресурс] - https://www.mathworks.com/products/matlab.html
16. Mathcad [Електронний ресурс] - https://www.mathcad.com/ru
17. Autocad [Електронний ресурс] - https://www.autodesk.com/products/autocad/overview?plc=ACDIST&term=1-YEAR&support=ADVANCED&quantity=1
18. C# [Електронний ресурс] - https://metanit.com/sharp/

ДОДАТОК А

Побудова мінімальних поверхонь на основі B-сплайнів з квазіконформною заміною параметру

Специфікація

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТВ6129_20Б

Аркушів 2

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ імені Ігоря Сікорського ”_ТЕФ_АПЕПС_ТВ6129_20Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ КПІ імені Ігоря Сікорського ”_ТЕФ_АПЕПС_ТВ6129_20Б 12-1	Models	Пакет, що містить класи, які реалізують необхідні алгоритми та виконують розрахунки для побудови мінімальних поверхонь
УКР.НТУУ”КПІ КПІ імені Ігоря Сікорського ”_ТЕФ_АПЕПС_ТВ6129_20Б 12-2	View	Пакет, що містить графічний інтерфейс
УКР.НТУУ”КПІ КПІ імені Ігоря Сікорського ”_ТЕФ_АПЕПС_ТВ6129_20Б 12-3	Controllers	Пакет, що містить контролер системи

ДОДАТОК Б

Побудова мінімальних поверхонь на основі B-сплайнів з квазіконформною заміною параметру

Текст програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТВ6129_20Б 12-1

Аркушів 14

Київ 2020

```
using System.Collections.Generic;

using BSplineGridWebApp.Models.BusinessLogic.Abstractions;


namespace BSplineGridWebApp.Models.BusinessLogic.BSpline

{

   public class BSplineBuilder

   {

      public IBasicFunctionExecutor BasicFunctionExecutor { get; set; }

      public  IEnumerable<global::BSplineGridWebApp.Models.BusinessLogic.Point.Point>  ControlPoints  { get;
set; }

      public int Order { get; set; }

      public double[] NodalVector { get; set; }

      public global::BSplineGridWebApp.Models.BusinessLogic.Point.Point BSplinePoint { get; set; }




      public                   BSplineBuilder(IBasicFunctionExecutor                   basicFunctionExecutor,
IEnumerable<global::BSplineGridWebApp.Models.BusinessLogic.Point.Point> controlPoints,

         int order, double[] nodalVector)

      {

         BasicFunctionExecutor = basicFunctionExecutor;

         ControlPoints = controlPoints;

         Order = order;

         NodalVector = nodalVector;

      }


      public global::BSplineGridWebApp.Models.BusinessLogic.Point.Point Execute(ComplexBaseArgument t)

      {

         BSplinePoint       =       new       global::BSplineGridWebApp.Models.BusinessLogic.Point.Point(new
ComplexBaseArgument(0,0),

            new ComplexBaseArgument(0,0),

            new ComplexBaseArgument(0,0));

         int indexOfBasicFunction = 0;

         foreach (var controlPoint in ControlPoints)

         {

            ComplexBaseArgument valueOfBasicFunc =

               BasicFunctionExecutor.GetValueOfBasicFunc(Order, indexOfBasicFunction, NodalVector, t);

            BSplinePoint.X += controlPoint.X * valueOfBasicFunc;

            BSplinePoint.Y += controlPoint.Y * valueOfBasicFunc;
```

```
            BSplinePoint.Z += controlPoint.Z * valueOfBasicFunc;
            indexOfBasicFunction++;
         }

         return BSplinePoint;
      }
   }
}

using System;
using System.Linq;
using BSplineGridWebApp.Models.BusinessLogic.Abstractions;

namespace BSplineGridWebApp.Models.BusinessLogic.BasicFunctionExecutor
{
   public class BasicFunctionExecutor : IBasicFunctionExecutor
   {
      public ComplexBaseArgument GetValueOfBasicFunc (int order, int indexOfBasicFunction, double[]
nodalVector,
         ComplexBaseArgument x)
      {
         ComplexBaseArgument result;

       if (order == 0)
         {
            if (x >= nodalVector.ElementAt(indexOfBasicFunction) &&
               x < nodalVector.ElementAt(indexOfBasicFunction + 1))
            {
               result = new ComplexBaseArgument(1,0);
            }

            else
            {
           result =  new ComplexBaseArgument(0,0);
            }

         }
```

```
        else
        {
            double firstDenominator, secondDenominator;
            ComplexBaseArgument firstMultiplier, secondMultiplier;

            firstDenominator = nodalVector[indexOfBasicFunction + order] -
nodalVector[indexOfBasicFunction];
            secondDenominator = nodalVector[indexOfBasicFunction + order + 1] -
                        nodalVector[indexOfBasicFunction + 1];

            if (Math.Abs(firstDenominator) < 0.0000001)
            {
                firstMultiplier = new ComplexBaseArgument(0,0);
            }
            else
            {
                firstMultiplier = (x - nodalVector[indexOfBasicFunction]) / firstDenominator;
            }

            if (Math.Abs(secondDenominator) < 0.0000001)
            {
                secondMultiplier = new ComplexBaseArgument(0,0);
            }
            else
            {
                secondMultiplier = (nodalVector[indexOfBasicFunction + order + 1] - x) / secondDenominator;
            }

            result = firstMultiplier * GetValueOfBasicFunc(order - 1, indexOfBasicFunction, nodalVector, x)
                + secondMultiplier * GetValueOfBasicFunc(order - 1, indexOfBasicFunction + 1,
nodalVector, x);
        }

        return result;
    }
}
```

```
}

using BSplineGridWebApp.Models.BusinessLogic.Abstractions;


namespace BSplineGridWebApp.Models.BusinessLogic.Point
{
   public class Point
   {
      public ComplexBaseArgument X { get; set; }


      public ComplexBaseArgument Y { get; set; }


      public ComplexBaseArgument Z { get; set; }


      public Point(ComplexBaseArgument x, ComplexBaseArgument y, ComplexBaseArgument z)
      {
         X = x;
         Y = y;
         Z = z;
      }
   }
}


namespace BSplineGridWebApp.Models.BusinessLogic.Abstractions
{
   public interface IBasicFunctionExecutor
   {
      public ComplexBaseArgument GetValueOfBasicFunc(int order, int indexOfBasicFunction, double[]
nodalVector,
         ComplexBaseArgument x);
    }
}

using System;


namespace BSplineGridWebApp.Models.BusinessLogic.Abstractions
{
   public class ComplexBaseArgument
```

```
{
    public double RealPart { get; set; }

    public double ImaginePart { get; set; }

    public ComplexBaseArgument(double realPart, double imaginePart)
    {
        RealPart = realPart;
        ImaginePart = imaginePart;
    }

    public static bool operator >(ComplexBaseArgument complexArg, double second)
    {
        return complexArg.RealPart > second;
    }

    public static bool operator <(ComplexBaseArgument complexArg, double second)
    {
        return complexArg.RealPart < second;
    }

    public static bool operator >= (ComplexBaseArgument complexArg, double second)
    {
        return complexArg.RealPart >= second;
    }

    public static bool operator <= (ComplexBaseArgument complexArg, double second)
    {
        return complexArg.RealPart <= second;
    }

    public static bool operator >(double second, ComplexBaseArgument complexArg)
    {
        return second > complexArg.RealPart;
    }
```

```
public static bool operator <(double second, ComplexBaseArgument complexArg)
{
   return second < complexArg.RealPart;
}

public static bool operator >= (double second, ComplexBaseArgument complexArg)
{
   return second >= complexArg.RealPart;
}

public static bool operator <= (double second, ComplexBaseArgument complexArg)
{
   return second <= complexArg.RealPart;
}

public static ComplexBaseArgument operator +(ComplexBaseArgument first, ComplexBaseArgument second)
{
   return new ComplexBaseArgument(realPart: first.RealPart + second.RealPart,
      imaginePart: first.ImaginePart + second.ImaginePart);
}

public static ComplexBaseArgument operator -(ComplexBaseArgument first, ComplexBaseArgument second)
{
   return new ComplexBaseArgument(realPart: first.RealPart - second.RealPart,
      imaginePart: first.ImaginePart - second.ImaginePart);
}

public static ComplexBaseArgument operator -(ComplexBaseArgument complexArg)
{
   return new ComplexBaseArgument(0.0-complexArg.RealPart, 0.0-complexArg.ImaginePart);
}

public static ComplexBaseArgument operator *(ComplexBaseArgument first, ComplexBaseArgument second )
{
   return new ComplexBaseArgument(
      first.RealPart * second.RealPart - first.ImaginePart * second.ImaginePart,
```

```
            first.RealPart * second.ImaginePart + first.ImaginePart * second.RealPart);
        }

        public static ComplexBaseArgument operator *(double doubleArg, ComplexBaseArgument complexArg)
        {
            return new ComplexBaseArgument(complexArg.RealPart * doubleArg, complexArg.ImaginePart * doubleArg);
        }

        public static ComplexBaseArgument operator *(ComplexBaseArgument complexArg, double doubleArg)
        {
            return new ComplexBaseArgument(complexArg.RealPart * doubleArg, complexArg.ImaginePart * doubleArg);
        }

        public static ComplexBaseArgument operator -(double doubleArg, ComplexBaseArgument complexArg)
        {
            return new ComplexBaseArgument(doubleArg - complexArg.RealPart, -complexArg.ImaginePart);
        }

        public static ComplexBaseArgument operator -(ComplexBaseArgument complexArg, double doubleArg)
        {
            return new ComplexBaseArgument(complexArg.RealPart - doubleArg, complexArg.ImaginePart);
        }

        public static ComplexBaseArgument operator +(double doubleArg, ComplexBaseArgument complexArg)
        {
            return new ComplexBaseArgument(doubleArg + complexArg.RealPart, complexArg.ImaginePart);
        }

        public static ComplexBaseArgument operator +(ComplexBaseArgument complexArg, double doubleArg)
        {
            return new ComplexBaseArgument(complexArg.RealPart + doubleArg, complexArg.ImaginePart);
        }

        public static ComplexBaseArgument operator ^(ComplexBaseArgument complexArg, int power)
        {
```

```
            double moduleComplexNumber = Math.Sqrt(complexArg.RealPart * complexArg.RealPart
                                        + complexArg.ImaginePart * complexArg.ImaginePart);


            double angle = Math.Atan(complexArg.ImaginePart / complexArg.RealPart);



            return (Math.Pow(moduleComplexNumber, power))*(new ComplexBaseArgument( Math.Cos(power * angle),
Math.Sin(power * angle)));
        }


        public static ComplexBaseArgument operator /(ComplexBaseArgument complexArg, double doubleArg)
        {
            return new ComplexBaseArgument((complexArg.RealPart*doubleArg)/(Math.Pow(doubleArg,2)),
                (complexArg.ImaginePart * doubleArg)/(Math.Pow(doubleArg, 2)));
        }


        public static ComplexBaseArgument operator /(ComplexBaseArgument first, ComplexBaseArgument second)
        {
            return new ComplexBaseArgument(0,0)
            {
                RealPart                    =                    (first.RealPart*second.RealPart                    +
first.ImaginePart*second.ImaginePart)/(Math.Pow(second.RealPart, 2) + Math.Pow(second.ImaginePart, 2)),
                ImaginePart                 =                 (first.ImaginePart*second.RealPart                 -
first.RealPart*second.ImaginePart)/(Math.Pow(second.RealPart, 2) + Math.Pow(second.ImaginePart, 2))
            };
        }


        public override string ToString() {
            return $"{RealPart} + {ImaginePart} * i";
        }
    }
}


using System.Collections.Generic;
using System.Linq;


namespace BSplineGridWebApp.Models.BusinessLogic.Helpers
{
```

```
    public class ImaginePartDeterminant

    {

        public static void DefineImaginePart(IList<global::BSplineGridWebApp.Models.BusinessLogic.Point.Point> points)

        {

            for (int i = 0; i <= points.Count - 2; i++)

            {

                points.ElementAt(i + 1).X.ImaginePart =

                    - points.ElementAt(i + 1).Y.RealPart + points.ElementAt(i).X.ImaginePart + points.ElementAt(i).Y.RealPart;



                points.ElementAt(i + 1).Y.ImaginePart =

                    points.ElementAt(i + 1).X.RealPart - points.ElementAt(i).X.RealPart

                    + points.ElementAt(i).Y.ImaginePart;

            }

        }

    }}

using System.Collections.Generic;

using BSplineGridWebApp.Models.BusinessLogic.Abstractions;



namespace BSplineGridWebApp.Models.BusinessLogic.Helpers

{

    public class PrivateDerivativeHelper

    {

        public static List<double> ExecuteDerivativeXByU(ComplexBaseArgument[,] arguments, global::BSplineGridWebApp.Models.BusinessLogic.Point.Point[,] valueOfFunc, double step)

        {

            List<double> result = new List<double>();



            for (int i = 0; i < 1; i++)

            {

                for (int j = 0; j < 1; j++)

                {

                    result.Add((valueOfFunc[i+1,j].X.RealPart - valueOfFunc[i,j].X.RealPart) / step);

                }

            }



            for (int i = 1; i < arguments.GetLength(0) - 2; i++)
```

```
        {
            for (int j = 1; j < arguments.GetLength(1) - 2; j++)
            {
                result.Add((valueOfFunc[i+1,j].X.RealPart - valueOfFunc[i - 1,j].X.RealPart) / (2*step));
            }
        }

        return result;
    }

    public static List<double> ExecuteDerivativeXByV(ComplexBaseArgument[,] arguments,
global::BSplineGridWebApp.Models.BusinessLogic.Point.Point[,] valueOfFunc, double step)
    {
        List<double> result = new List<double>();

        for (int i = 0; i < 1; i++)
        {
            for (int j = 0; j < 1; j++)
            {
                result.Add((valueOfFunc[i, j + 1].X.RealPart - valueOfFunc[i,j].X.RealPart) / step);
            }
        }

        for (int i = 1; i < arguments.GetLength(0) - 2; i++)
        {
            for (int j = 1; j < arguments.GetLength(1) - 2; j++)
            {
                result.Add((valueOfFunc[i, j + 1].X.RealPart - valueOfFunc[i,j - 1].X.RealPart) / (2*step));
            }
        }

        return result;
    }

    public static List<double> ExecuteDerivativeYByU(ComplexBaseArgument[,] arguments,
global::BSplineGridWebApp.Models.BusinessLogic.Point.Point[,] valueOfFunc, double step)
    {
```

```
        List<double> result = new List<double>();

        for (int i = 0; i < 1; i++)
        {
            for (int j = 0; j < 1; j++)
            {
                result.Add((valueOfFunc[i+1,j].Y.RealPart - valueOfFunc[i,j].Y.RealPart) / step);
            }
        }

        for (int i = 1; i < arguments.GetLength(0) - 2; i++)
        {
            for (int j = 1; j < arguments.GetLength(1) - 2; j++)
            {
                result.Add((valueOfFunc[i + 1, j].Y.RealPart - valueOfFunc[i - 1,j].Y.RealPart) / (2*step));
            }
        }

        return result;
    }

    public static List<double> ExecuteDerivativeYByV(ComplexBaseArgument[,] arguments,
global::BSplineGridWebApp.Models.BusinessLogic.Point.Point[,] valueOfFunc, double step)
    {
        List<double> result = new List<double>();

        for (int i = 0; i < 1; i++)
        {
            for (int j = 0; j < 1; j++)
            {
                result.Add((valueOfFunc[i, j + 1].Y.RealPart - valueOfFunc[i,j].Y.RealPart) / step);
            }
        }

        for (int i = 1; i < arguments.GetLength(0) - 2; i++)
        {
            for (int j = 1; j < arguments.GetLength(1) - 2; j++)
```

```
            {
                result.Add((valueOfFunc[i, j + 1].Y.RealPart - valueOfFunc[i,j - 1].Y.RealPart) / (2*step));
            }
        }

        return result;
    }

    public static List<double> ExecuteDerivativeZByU(ComplexBaseArgument[,] arguments,
global::BSplineGridWebApp.Models.BusinessLogic.Point.Point[,] valueOfFunc, double step)
    {
        List<double> result = new List<double>();

        for (int i = 0; i < 1; i++)
        {
            for (int j = 0; j < 1; j++)
            {
                result.Add((valueOfFunc[i+1,j].Z.RealPart - valueOfFunc[i,j].Z.RealPart) / step);
            }
        }

        for (int i = 1; i < arguments.GetLength(0) - 2; i++)
        {
            for (int j = 1; j < arguments.GetLength(1) - 2; j++)
            {
                result.Add((valueOfFunc[i+1,j].Z.RealPart - valueOfFunc[i - 1,j].Z.RealPart) / (2*step));
            }
        }

        return result;
    }

    public static List<double> ExecuteDerivativeZByV(ComplexBaseArgument[,] arguments,
global::BSplineGridWebApp.Models.BusinessLogic.Point.Point[,] valueOfFunc, double step)
    {
        List<double> result = new List<double>();
```

```
            for (int i = 0; i < 1; i++)
            {
                for (int j = 0; j < 1; j++)
                {
                    result.Add((valueOfFunc[i, j + 1].Z.RealPart - valueOfFunc[i,j].Z.RealPart) / step);
                }
            }

            for (int i = 1; i < arguments.GetLength(0) - 2; i++)
            {
                for (int j = 1; j < arguments.GetLength(1) - 2; j++)
                {
                    result.Add((valueOfFunc[i, j + 1].Z.RealPart - valueOfFunc[i,j - 1].Z.RealPart) / (2*step));
                }
            }

            return result;
        }
    }
}
```

ДОДАТОК В

Побудова мінімальних поверхонь на основі B-сплайнів з квазіконформною заміною параметру

Опис програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТВ6129_20Б 13-1

Аркушів 9

Київ 2020

АНОТАЦІЯ

Розроблена система містить три компоненти. Перший компонент - це пакет Models, що містить класи та інтерфейси, що описують предметну область та реалізують необхідні алгоритми і виконують геометричні розрахунки, математичні розрахунки, використовуючи алгебру комплексних чисел для побудови мінімальних поверхонь на основі B-сплайнів з квазіконформною заміною параметру.

Другий компонент - це пакет Views, що містить реалізацію графічного інтерфейсу системи, який візуалізую дискретні значення координат точок мінімальної поверхні, що були розраховані за допомогою бізнес-логіки системи та надає користувачу додаткові можливості для дослідження поверхні, а саме: обертання поверхні під різними кутами, збільшення ділянки поверхні, перегляд значення координат поверхні тощо, а також надає для перегляду таблицю зі значеннями характеристичних коефіцієнтів для точок мінімальної поверхні.

Третій компонент - пакет Controllers, що містить клас-контролер системи, який пов’язує бізнес-логіку системи з графічним представленням, оброблює запити клієнта, валідує вхідну початкову інформацію, є точкою входу в систему.

Програмне забезпечення було створене в інтегрованому середовищі розробки Rider 2020 об’єктно-орієнтованою мовою програмування C#, використовуючи технологію ASP.NET Core MVC. Для графічного інтерфейсу було використано мову розмітку html, таблицю каскадних стилів css, мову програмування js та графічну бібліотеку plotly.js.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Розроблений програмний продукт працює на базі таких операційних системи як Windows, Linux, macOs. Для функціонування потребує попереднього встановлення пакету asp.net core ryntime та будь-якого веб-браузеру з підтримкою js.

Програмний продукт розроблений об’єктно-орієнтованою мовою програмування C# в інтегрованому середовищі розробки Rider 2020, використовуючи технологію ASP.NET Core MVC. Для графічного інтерфейсу було використано мову розмітку html, таблицю каскадних стилів css, мову програмування js та графічну бібліотеку plotly.js.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Програмна система вирішує завдання побудови мінімальних поверхнонь на основі B-сплайнів з квазіконформною заміною параметру.

Програмна система призначена для побудови мінімальних поверхнонь на основі B-сплайнів з квазіконформною заміною параметру, базуючись на початковій вхідній інформації, яку надава користувач.

Функціональних обмежень на використання розроблених компонентів не має.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Перед початком побудови мінімальної поверхні, користувачеві необхідно завантажити у систему попередньо створений та заповнений файл з розширенням .json, що містить необхідну початкову інформацію про поверхню як значення комплексних координат точок характеристичного чотирикутника та вузлового вектору.

Після цього, програма, отримавши цей файл, зчитаваши та провалідувавши, може розраховувати дискретні значення координат точок мінімальної поверхні та графічно представляти у вигляді поверхні.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Розроблений програмний продукт працює на базі платформи .Net Core та таких операційних систем як Windows, Linux та macOS. Система потребує встановлення пакету asp.net core runtime та будь-якого веб-браузеру з підтримкою js.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблений програмний додаток не потрібно встановлювати на персональний комп’ютер. Для роботи з ним необхідно просто завантажити заархівований файл, розархівувати його та перейти у папку за наступною адресою, а саме: BSplineMinimalSurfaceWebApp/BSplineGridWebApp/BSplineGridWebApp/, відкрити вікно терміналу в ній та ввести команду dotnet run.

Система потребує потребує встановлення пакету asp.net core runtime та будь-якого веб-браузеру з підтримкою js., займає невеликий обсяг пам’яті та має невеликі системні вимоги. Для видалення програми з комп’ютера достатньо видалити папку, що містить файли проекту.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для програмної системи є файл з розширенням .json, що містить початкову вхідну інформацію, а саме комплексні значення координат точок характеристичного чотирикутника та вузлового вектору.

Вихідними даними є графічне представлення дискретних значень координат точок поверхні та таблиця зі значеннями характеристичних коефіцієнтів для точок поверхні.